

Included Refereed publications in chronological order

- [1] Hajime Ishihara, Bakhadyr Khoussainov, and Sasha Rubin. “Some Results on Automatic Structures”. In: *LICS 2002, 17th IEEE Symposium on Logic in Computer Science, 22-25 July 2002, Copenhagen, Denmark, Proceedings*. 2002, p. 235.
- [2] Bakhadyr Khoussainov, Sasha Rubin, and Frank Stephan. “On Automatic Partial Orders”. In: *LICS 2003, 18th IEEE Symposium on Logic in Computer Science, 22-25 June 2003, Ottawa, Canada, Proceedings*. 2003, pp. 168–177.
- [3] Doron Bustan, Sasha Rubin, and Moshe Y. Vardi. “Verifying omega-Regular Properties of Markov Chains”. In: *Computer Aided Verification, 16th International Conference, CAV 2004, Boston, MA, USA, July 13-17, 2004, Proceedings*. 2004, pp. 189–201.
- [4] Bakhadyr Khoussainov, André Nies, Sasha Rubin, and Frank Stephan. “Automatic Structures: Richness and Limitations”. In: *LICS 2004, 19th IEEE Symposium on Logic in Computer Science, 14-17 July 2004, Turku, Finland, Proceedings*. 2004, pp. 44–53.
- [5] Bakhadyr Khoussainov, Sasha Rubin, and Frank Stephan. “Definability and Regularity in Automatic Structures”. In: *STACS 2004, 21st Annual Symposium on Theoretical Aspects of Computer Science, Montpellier, France, March 25-27, 2004, Proceedings*. 2004, pp. 440–451.
- [6] Bakhadyr Khoussainov and Sasha Rubin. “Decidability of Term Algebras Extending Partial Algebras”. In: *Computer Science Logic, 19th International Workshop, CSL 2005, 14th Annual Conference of the EACSL, Oxford, UK, August 22-25, 2005, Proceedings*. 2005, pp. 292–308.
- [7] Bakhadyr Khoussainov, Sasha Rubin, and Frank Stephan. “Automatic linear orders and trees”. In: *ACM Trans. Comput. Log.* 6.4 (2005), pp. 675–700.
- [8] Bakhadyr Khoussainov, André Nies, Sasha Rubin, and Frank Stephan. “Automatic Structures: Richness and Limitations”. In: *Logical Methods in Computer Science* 3.2 (2007).
- [9] Tobias Ganzow and Sasha Rubin. “Order-Invariant MSO is Stronger than Counting MSO in the Finite”. In: *STACS 2008, 25th Annual Symposium on Theoretical Aspects of Computer Science, Bordeaux, France, February 21-23, 2008, Proceedings*. 2008, pp. 313–324.
- [10] Lukasz Kaiser, Sasha Rubin, and Vince Bárány. “Cardinality and counting quantifiers on omega-automatic structures”. In: *STACS 2008, 25th Annual Symposium on Theoretical Aspects of Computer Science, Bordeaux, France, February 21-23, 2008, Proceedings*. 2008, pp. 385–396.
- [11] Sasha Rubin. “Automata Presenting Structures: A Survey of the Finite String Case”. In: *Bulletin of Symbolic Logic* 14.2 (2008), pp. 169–209.

- [12] Vince Bárány, Erich Grädel, and Sasha Rubin. “Automata-based presentations of infinite structures”. In: *Finite and Algorithmic Model Theory*. Ed. by Javier Esparza, Christian Michaux, and Charles Steinhorn. Cambridge Books Online. Cambridge University Press, 2011, pp. 1–76.
- [13] Alex Kruckman, Sasha Rubin, John Sheridan, and Ben Zax. “A Myhill-Nerode theorem for automata with advice”. In: *Proceedings Third International Symposium on Games, Automata, Logics and Formal Verification, GandALF 2012, Napoli, Italy, September 6-8, 2012*. 2012, pp. 238–246.
- [14] Alexander Rabinovich and Sasha Rubin. “Interpretations in Trees with Countably Many Branches”. In: *LICS 2012, Proceedings of the 27th Annual IEEE Symposium on Logic in Computer Science, Dubrovnik, Croatia, June 25-28, 2012*. 2012, pp. 551–560.
- [15] Krishnendu Chatterjee, Siddhesh Chaubal, and Sasha Rubin. “How to Travel between Languages”. In: *Language and Automata Theory and Applications - 7th International Conference, LATA 2013, Bilbao, Spain, April 2-5, 2013. Proceedings*. 2013, pp. 214–225.
- [16] Benjamin Aminof, Swen Jacobs, Ayrat Khalimov, and Sasha Rubin. “Parameterized Model Checking of Token-Passing Systems”. In: *Verification, Model Checking, and Abstract Interpretation - 15th International Conference, VMCAI 2014, San Diego, CA, USA, January 19-21, 2014, Proceedings*. 2014, pp. 262–281.
- [17] Benjamin Aminof, Tomer Kotek, Sasha Rubin, Francesco Spegni, and Helmut Veith. “Parameterized Model Checking of Rendezvous Systems”. In: *CONCUR 2014 - Concurrency Theory - 25th International Conference, CONCUR 2014, Rome, Italy, September 2-5, 2014. Proceedings*. 2014, pp. 109–124.
- [18] Benjamin Aminof and Sasha Rubin. “First Cycle Games”. In: *Proceedings 2nd International Workshop on Strategic Reasoning, SR 2014, Grenoble, France, April 5-6, 2014*. 2014, pp. 83–90.
- [19] Andrey Grinshpun, Pakawat Phalitnonkiat, Sasha Rubin, and Andrei Tarfulea. “Alternating traps in Muller and parity games”. In: *Theor. Comput. Sci.* 521 (2014), pp. 73–91.
- [20] Benjamin Aminof, Aniello Murano, and Sasha Rubin. “On CTL* with Graded Path Modalities”. In: *Logic for Programming, Artificial Intelligence, and Reasoning - 20th International Conference, LPAR-20 2015, Suva, Fiji, November 24-28, 2015, Proceedings*. 2015, pp. 281–296.
- [21] Benjamin Aminof, Aniello Murano, Sasha Rubin, and Florian Zuleger. “Verification of Asynchronous Mobile-Robots in Partially-Known Environments”. In: *PRIMA 2015: Principles and Practice of Multi-Agent Systems - 18th International Conference, Bertinoro, Italy, October 26-30, 2015, Proceedings*. 2015, pp. 185–200.

- [22] Benjamin Aminof, Sasha Rubin, Francesco Sogno, and Florian Zuleger. “Liveness of Parameterized Timed Networks”. In: *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part II*. 2015, pp. 375–387.
- [23] Benjamin Aminof, Sasha Rubin, and Florian Zuleger. “On the Expressive Power of Communication Primitives in Parameterised Systems”. In: *Logic for Programming, Artificial Intelligence, and Reasoning - 20th International Conference, LPAR-20 2015, Suva, Fiji, November 24-28, 2015, Proceedings*. 2015, pp. 313–328.
- [24] Roderick Bloem, Swen Jacobs, Ayrat Khalimov, Igor Konnov, Sasha Rubin, Helmut Veith, and Josef Widder. *Decidability of Parameterized Verification*. Synthesis Lectures on Distributed Computing Theory. Morgan & Claypool Publishers, 2015.
- [25] Aniello Murano, Giuseppe Perelli, and Sasha Rubin. “Multi-agent Path Planning in Known Dynamic Environments”. In: *PRIMA 2015: Principles and Practice of Multi-Agent Systems - 18th International Conference, Bertinoro, Italy, October 26-30, 2015, Proceedings*. 2015, pp. 218–231.
- [26] Sasha Rubin. “Parameterised Verification of Autonomous Mobile-Agents in Static but Unknown Environments”. In: *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2015, Istanbul, Turkey, May 4-8, 2015*. 2015, pp. 199–208.
- [27] Benjamin Aminof, Vadim Malvone, Aniello Murano, and Sasha Rubin. “Graded Strategy Logic”. In: *Proceedings 4th International Workshop on Strategic Reasoning, SR 2016, New York, USA*. 2016.
- [28] Benjamin Aminof, Vadim Malvone, Aniello Murano, and Sasha Rubin. “Graded Strategy Logic: Reasoning about Uniqueness of Nash Equilibria”. In: *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems, Singapore, May 9-13, 2016*. 2016, pp. 698–706.
- [29] Benjamin Aminof, Aniello Murano, Sasha Rubin, and Florian Zuleger. “Automatic Verification of Multi-Agent Systems in Parameterised Grid-Environments”. In: *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems, Singapore, May 9-13, 2016*. 2016, pp. 1190–1199.
- [30] Benjamin Aminof, Aniello Murano, Sasha Rubin, and Florian Zuleger. “Prompt Alternating-Time Epistemic Logics”. In: *Principles of Knowledge Representation and Reasoning: Proceedings of the Fifteenth International Conference, KR 2016, Cape Town, South Africa, April 25-29, 2016*. 2016, pp. 258–267.
- [31] Benjamin Aminof and Sasha Rubin. “First Cycle Games”. In: *Information and Computation*, (2016). DOI: <http://dx.doi.org/10.1016/j.ic.2016.10.008>.

- [32] Benjamin Aminof and Sasha Rubin. “Model Checking Parameterised Multi-token Systems via the Composition Method”. In: *Automated Reasoning - 8th International Joint Conference, IJCAR 2016, Coimbra, Portugal, June 27 - July 2, 2016, Proceedings*. 2016, pp. 499–515.
- [33] Roderick Bloem, Swen Jacobs, Ayrat Khalimov, Igor Konnov, Sasha Rubin, Helmut Veith, and Josef Widder. “Decidability in Parameterized Verification”. In: *SIGACT News* 47.2 (2016), pp. 53–64.
- [34] Giuseppe De Giacomo, Antonio Di Stasio, Aniello Murano, and Sasha Rubin. “Imperfect information games and generalized planning”. In: *International Joint Conference on Artificial Intelligence (IJCAI 2016)*. 2016.
- [35] Francesco Belardinelli, Alessio Lomuscio, Aniello Murano, and Sasha Rubin. “Verification of Multi-agent Systems with Imperfect Information and Public Actions”. In: *Proceedings of the 2017 International Conference on Autonomous Agents & Multiagent Systems, São Paulo, May 8-12, 2017*. 2017.
- [36] Raphael Berthon, Bastien Maubert, Aniello Murano, Sasha Rubin, and Moshe Vardi. “Hierarchical Strategic Reasoning”. In: *LICS 2017*. 2017.

Some results on automatic structures

Hajime Ishihara
School of Information Sciences
JAIST, Japan
ishihara@jaist.ac.jp

Bakhadyr Khoussainov
Computer Science Department
The University of Auckland, New Zealand
bmk@cs.auckland.ac.nz

Sasha Rubin
Mathematics Department
The University of Auckland, New Zealand
rubin@math.auckland.ac.nz

Abstract

We study the class of countable structures which can be presented by synchronous finite automata. We reduce the problem of existence of an automatic presentation of a structure to that for a graph. We exhibit a series of properties of automatic equivalence structures, linearly ordered sets and permutation structures. These serve as a first step in producing practical descriptions of some automatic structures or illuminating the complexity of doing so for others.

1. Introduction

In this paper we investigate those countable structures that can be presented, in a certain precise sense, by means of finite automata. The general idea is to code elements of a given structure in such away that all the atomic first order queries about the structure can be decided by finite automata. We call these structures automatic structures. In this paper we will be interested in the classes of automatic graphs, equivalence structures, linear orderings and permutation structures. Our basic motivation lies in trying to characterise, in an appropriate terminology, isomorphism invariants of automatic structures. Here by isomorphism invariant of a structure we mean a property of the structure expressible in a certain formalism – say, first or second order logic. If such a property is preserved under isomorphisms of the structure, we call it an isomorphism invariant. For example if the structure in question is an equivalence relation, then the number of equivalence classes of any given size is an isomorphism invariant. In [5] Blumensath and Grädel characterised automatic structures in terms of an important concept of logic, namely interpretability. They proved that

a structure \mathcal{A} is automatic if and only if it is first order interpretable in the structure $N_p = (\mathbb{N}, +, |_p)$, where $x|_p y$ iff $x = p^n$ and $y = kx$ for some $n, k \in \mathbb{N}$. However, it seems that the problem of characterising the isomorphism invariants of automatic structures is a challenging task. We will show that even for simple cases such as equivalence structures and permutation structures the situation is quite complex.

There are several reasons to be interested in understanding isomorphism invariants of automatic structures. One is that we would like to understand the interplay between automata-theoretic and model-theoretic (or algebraic) concepts, e.g. recognisability and definability. The other reason is of complexity-theoretic nature. It is known that the first order theory of any automatic structure is decidable [11]. A natural question arises as to which automatic structures are feasible and which are not. Blumensath and Grädel [4] [5] [9] show that there are automatic structures whose theories are non-elementary. In other words, the expression complexity of the model checking problem in automatic structures – that is, the problem of checking whether a formula is satisfied in a given structure – can be intractable. On the other hand, there are examples of automatic structures, e.g. structures presented by finite automata over unary alphabets or the rational numbers with the natural ordering, for which the expression complexity of the model checking problem is polynomial. These and other results on automatic structures implicitly tell us that there are intimate interactions between studying isomorphism invariants of automatic structures and the expression complexity of the model checking problem. The more we know about isomorphism invariants of automatic structures, the more the theory of this structure is computationally accessible.

We now give an overview of this paper. The next section is an introductory section where we give basic defini-

tions and state some known results in the area. The section on automatic graphs is devoted to exhibiting a functor from the class of all automatic structures into the class of all automatic graphs. We prove that this functor preserves not only model-theoretic but also automata-theoretic properties of structures. The section on automatic equivalence structures is devoted to constructing automatic equivalence relations with different types of isomorphism invariants. The main goal is to show that some isomorphism invariants of automatic equivalence structures possess complicated complexity-theoretic and algebraic behaviour. The next section reduces the study of automatic equivalence relations to that of automatic linear orderings. Finally, in the last section we study automatic permutations. We construct automatic permutations from automatic equivalence structures. We also show the relationship between permutation structures and the running times of reversible Turing machines. As a consequence of this relationship we will prove that the isomorphism problem for permutation structures is undecidable.

We briefly note related work. A systematic study of interactions between automata and algebraic structures began from the work of Cannon and Thurston on automatic groups [8]. This was generalised by Khoussainov and Nerode in [11] but motivated from a point of view of computable model theory. A significant work in the understanding of automatic structures is due to Blumensath and Grädel [4] [5]. A recent paper by Benedikt and et al. [1] investigates model-theoretic properties of automatic structures, e.g. questions related to quantifier elimination. In unpublished work Delhomme, et al. [6] show that the minimal ordinal without an automatic presentation is ω^ω .

2. Basic Notions

A *finite automaton (FA)* \mathcal{A} over an alphabet Σ is a tuple (S, I, Δ, F) , where S is a finite set of *states*, $I \subset S$ is the set of *initial states*, $\Delta \subset S \times \Sigma \times S$ is the *transition table* and $F \subset S$ is the set of *final states*. A *computation* of \mathcal{A} on a word $\sigma_1\sigma_2\dots\sigma_n$ ($\sigma_i \in \Sigma$) is a sequence of states, say q_0, q_1, \dots, q_n , such that $q_0 \in I$ and $(q_i, \sigma_{i+1}, q_{i+1}) \in \Delta$ for all $0 \leq i \leq n-1$. If $q_n \in F$, then the computation is *successful* and we say that automaton \mathcal{A} *accepts* the word. The *language*, $\mathcal{L}(\mathcal{A}) \subset \Sigma^*$, accepted by the automaton \mathcal{A} is the set of all words accepted by \mathcal{A} . In general, $D \subset \Sigma^*$ is *finite automaton (FA) recognisable*, or *regular*, if $D = \mathcal{L}(\mathcal{A})$ for some finite automaton \mathcal{A} . We assume that the reader is familiar with the standard basics in finite automata theory.

We now introduce *synchronous n-tape automata* which recognise n -ary relations. The following description is based on Eilenberg et al. [7]. A synchronous n -tape automaton can be thought of as a one-way Turing machine

with n input tapes. Each tape is regarded as semi-infinite having written on it a word in the alphabet Σ followed by an infinite succession of blanks, \diamond symbols. The automaton starts in an initial state, reads simultaneously the first symbol of each tape, changes state, reads simultaneously the second symbol of each tape, changes state, etc., until it reads a blank on each tape. The automaton then stops and accepts the n -tuple of words if it is in a final state. The set of all n -tuples accepted by the automaton is the relation recognised by the automaton.

Definition 1 Let Σ_\diamond be $\Sigma \cup \{\diamond\}$ where $\diamond \notin \Sigma$. The convolution of a tuple $(w_1, \dots, w_n) \in \Sigma^{*n}$ is the tuple $(w_1, \dots, w_n)^\diamond \in (\Sigma_\diamond)^{*n}$ formed by concatenating the least number of blank symbols, \diamond , to the right ends of the w_i , $1 \leq i \leq n$, so that the resulting words have equal length. The convolution of a relation $R \subset \Sigma^{*n}$ is the relation $R^\diamond \subset (\Sigma_\diamond)^{*n}$ formed as the set of convolutions of all the tuples in R .

Definition 2 An n -tape automaton on Σ is a finite automaton over the alphabet $(\Sigma_\diamond)^n$. An n -ary relation $R \subset \Sigma^{*n}$ is FA recognisable if its convolution R^\diamond is recognisable by an n -tape automaton.

We now relate n -tape automata to structures. A *structure* \mathcal{A} consists of a set A called the *domain* and some constants, relations and operations on A . We may assume that \mathcal{A} only contains relational and constant predicates as the operations can be replaced with their graphs. We write $\mathcal{A} = (A, R_1^A, \dots, R_k^A, c_0^A, \dots, c_t^A)$ where R_i^A is an n_i -ary relation on A and c_j^A is a constant element of A . Then $(R_1^{n_1}, \dots, R_k^{n_k}, c_0, \dots, c_t)$ is called the *signature* of \mathcal{A} . In the sequel, all structures are relational, have finite or countable domains and finite signatures.

Definition 3 A structure $\mathcal{A} = (A, R_1^A, \dots, R_k^A, c_0^A, \dots, c_t^A)$ is *automatic* over Σ if its domain $A \subset \Sigma^*$ and the relations $R_i^A \subset \Sigma^{*n_i}$ all are FA recognisable. An isomorphism from a structure \mathcal{B} to an automatic structure \mathcal{A} is an automatic presentation of \mathcal{B} in which case \mathcal{B} is called automatically presentable (over Σ). A structure will be called *automatic* if it is automatic over some alphabet.

The following result motivates studying automatic structures from a complexity-theoretic point of view.

Theorem 1 [11] There exists an algorithm that given an automatic structure \mathcal{A} and a first order definition of a relation R in \mathcal{A} produces a finite automaton that recognises R . In particular, the first order theory of \mathcal{A} is decidable. \square

Blumensath and Grädel extended this result in [4] by showing that the theorem holds even if one considers the

first order logic extended by the quantifier “there exist infinitely many”, denoted by $FO(\exists^\infty)$, and combined this with an important concept in model theory, namely interpretability.

Definition 4 Let \mathcal{A} and \mathcal{B} be structures of signatures L and K , respectively. An **n -dimensional interpretation** Γ of \mathcal{A} in \mathcal{B} consists of a $FO(\exists^\infty)$ -formula $\delta(x_1, \dots, x_n)$ of K , and for each symbol S of L a $FO(\exists^\infty)$ -formula $\phi_S(\bar{x}_1, \dots, \bar{x}_m)$ of K where each \bar{x}_i is an n -tuple of distinct variables and m is the arity of S , and a surjective map $f : \delta(B^n) \rightarrow A$ such that for all $\bar{b}_i \in \delta(B^n)$, $\mathcal{B} \models \phi_S(\bar{b}_1, \dots, \bar{b}_m) \iff (f\bar{b}_1, \dots, f\bar{b}_m) \in S^{\mathcal{A}}$.

We give an example. For a non-unary alphabet Σ consider the structures $\mathcal{W}(\Sigma) = (\Sigma^*, (\sigma_a)_{a \in \Sigma}, \preceq, el)$ and $N_p = (\mathbb{N}, +, |_p)$, where $\sigma_a(x) = xa$, $x \preceq y$ if x is a prefix of y , $el(x, y)$ if x and y have the same length, $x|_p y$ if x divides y and x is a power of p , and $+$ is addition. Then $N_{|\Sigma|}$ and $\mathcal{W}(\Sigma)$ are mutually interpretable. Here is an important theorem [5]:

Theorem 2 If \mathcal{B} is automatic and \mathcal{A} is interpretable in \mathcal{B} then \mathcal{A} is automatic.

3. On Automatic Graphs

In this section we provide a procedure that given an automatic structure \mathcal{A} produces an automatic graph $\mathcal{G}(\mathcal{A}) = (V(\mathcal{A}), E(\mathcal{A}))$, with vertices $V(\mathcal{A})$ and edges $E(\mathcal{A})$, so that \mathcal{A} and $\mathcal{G}(\mathcal{A})$ can be recovered from each other. The transformation of \mathcal{A} into $\mathcal{G}(\mathcal{A})$, denoted by Γ , is described in Hodges ([10] Theorem 5.5.1) and possesses natural algebraic and automata-theoretic properties. To investigate properties of Γ we need to explicitly define it with an eye towards automata-theoretic considerations.

An **n -tag**, where $n > 1$, is a symmetric graph isomorphic to the graph $(\{0, 1, \dots, n, c\}, E)$, where the set E of edges consists of all pairs $\{i, i+1\}$ for $0 \leq i < n$, $\{n, 1\}$ and $\{2, c\}$. The vertex 0 is the **start** of the n -tag. The element c is needed to make the tag rigid, that is a structure without nontrivial automorphisms. Furthermore c will not be mentioned explicitly.

Let v be a new symbol. With each element $a \in A$ we associate a 5-tag denoted by $T(a)$ so that the vertices of $T(a)$ are the words $va, va1, \dots, va5$ and edges $\{va, va1\}, \{vak, va(k+1)\}$ for $1 \leq k \leq 4$, and $\{va5, va1\}$. Here va is the start vertex of the tag $T(a)$, which we think of as representing the element a of the structure \mathcal{A} .

For $1 \leq i \leq n$, we code the predicate R_i of arity n_i as follows. Firstly, with each tuple $\bar{a} = (a_1, \dots, a_{n_i})$ for which $R_i(\bar{a})$ is true we associate a $(5+i)$ -tag $T(i, \bar{a})$ with vertices $v\bar{a}, v\bar{a}1, \dots, v\bar{a}(5+i)$ and edges $\{v\bar{a}, v\bar{a}1\}, \{v\bar{a}k, v\bar{a}(k+1)\}$ for $1 \leq k \leq 4$, and $\{v\bar{a}(5+i), v\bar{a}1\}$.

$\{v\bar{a}k, v\bar{a}(k+1)\}$ for $1 \leq k \leq i+4$, and $\{v\bar{a}(5+i), v\bar{a}\}$. Secondly, with each tuple $\bar{a} = (a_1, \dots, a_{n_i})$ for which $R_i(\bar{a})$ is true and where the k th element of this tuple is a_k , we associate the graph $L(i, \bar{a}, k)$ consisting of the k vertices $vak_1, v\bar{a}k_1, v\bar{a}k_2, v\bar{a}k_3, \dots, v\bar{a}kk, v\bar{a}$ and edges appearing between any consecutive pair in this list. Thus, $L(i, \bar{a}, k)$ establishes a path of length k between vak_1 and $v\bar{a}$ in case a_k is indeed the k th element of the tuple \bar{a} . The proof of the following lemma is left to the reader.

Lemma 1 If the domain A and the predicate R_i of the structure \mathcal{A} are regular language then:

1. The language $T(A) = \bigcup_{a \in A} T(a)$ and the binary relation $E_1(A) = \{(x, y) \mid \text{there is an } a \in A \text{ so that } \{x, y\} \text{ is an edge in } T(a)\}$ are regular.
2. The language $T(R_i) = \bigcup_{\bar{a} \in R_i} T(i, \bar{a})$ and the binary relation $E_2(R_i) = \{(x, y) \mid \text{there is an } \bar{a} \in R_i \text{ so that } \{x, y\} \text{ is an edge in } T(i, \bar{a})\}$ are regular.
3. The language $L(i) = \bigcup_{\bar{a} \in R_i, 1 \leq k \leq n_i} L(i, \bar{a}, k)$ and the binary relation $E_3(R_i) = \{(x, y) \mid \{x, y\} \text{ is an edge in some } L(i, \bar{a}, k)\}$ are regular. \square

Now define $\mathcal{G}(\mathcal{A}) = (V(\mathcal{A}), E(\mathcal{A}))$, where $V(\mathcal{A})$ is

$$T(A) \cup \bigcup_{1 \leq i \leq n} T(R_i) \cup \bigcup_{1 \leq i \leq n} L(i)$$

and $E(\mathcal{A})$ is

$$E_1(A) \cup \bigcup_{1 \leq i \leq n} E_2(R_i) \cup \bigcup_{1 \leq i \leq n} E_3(R_i).$$

Theorem 3 For the structure \mathcal{A} and the graph $\mathcal{G}(\mathcal{A})$ the following are true:

1. \mathcal{A} is automatic if and only if $\mathcal{G}(\mathcal{A})$ is automatic.
2. There is an isomorphism α between the group $Aut(\mathcal{A})$ of automorphisms of \mathcal{A} and the group $Aut(\mathcal{G}(\mathcal{A}))$ of automorphisms of $\mathcal{G}(\mathcal{A})$. Moreover, $f \in Aut(\mathcal{A})$ is automatic if and only if $\alpha(f)$ is automatic.
3. A substructure \mathcal{B} of \mathcal{A} is automatic if and only if the subgraph $\mathcal{G}(\mathcal{B})$ of $\mathcal{G}(\mathcal{A})$ is automatic.
4. A structure \mathcal{B} is automatically isomorphic to \mathcal{A} if and only if the graph $\mathcal{G}(\mathcal{B})$ is automatically isomorphic to $\mathcal{G}(\mathcal{A})$.
5. From any automatic presentation of \mathcal{A} an automatic presentation of $\mathcal{G}(\mathcal{A})$ can be constructed in linear time.

Proof. Part 1). Lemma 1 shows that if \mathcal{A} is automatic then so is $\mathcal{G}(\mathcal{A})$. Assume that $\mathcal{G}(\mathcal{A})$ is automatic. The set $D = \{x \mid x \text{ is the start of a 5-tag}\}$ is FA recognisable because it is FO-definable in $\mathcal{G}(\mathcal{A})$. For every $i, i = 1, \dots, n$, consider the relation $R_i = \{(x_1, \dots, x_n) \mid \text{there is an } x \text{ such that the distance between } x_k \text{ and } x \text{ is } k \text{ and } x \text{ is the start of a } 5+i\text{-tag}\}$. This relation is FO-definable and hence is FA recognisable. From the construction of $\mathcal{G}(\mathcal{A})$ we see that \mathcal{A} and (D, R_1, \dots, R_n) are isomorphic. Hence \mathcal{A} is automatic.

Part 2). Let f be an automorphism of \mathcal{A} . Define $\alpha(f) : \mathcal{G}(\mathcal{A}) \rightarrow \mathcal{G}(\mathcal{A})$ as follows. If $f(a) = b$ then set $\alpha(f)(va) = vb$. Take a tuple $\bar{a} = (a_1, \dots, a_n)$ so that $R_i(\bar{a})$ is true. Let $\bar{b} = (f(a_1), \dots, f(a_n))$. Set $\alpha(f)(v\bar{a}) = v\bar{b}$. Now extend this partial map to an automorphism $\alpha(f)$ of $\mathcal{G}(\mathcal{A})$. This automorphism is unique. The fact that α is an isomorphism can be checked by using the definition of $\mathcal{G}(\mathcal{A})$. Assume that f is an automatic automorphism of \mathcal{A} . We want to show that $\alpha(f)$ is an automatic automorphism of $\mathcal{G}(\mathcal{A})$. Take an $x \in V(\mathcal{A})$. Then either $x \in T(a)$ or $x \in T(i, \bar{a})$ or $x \in L(i, \bar{a}, k)$ for appropriate a, \bar{a}, i and k . Say, for instance $x \in T(a)$ and hence $x = vai$ for some $i = 0, \dots, 5$ (in case $i = 0$ we assume that $va0$ is va). From the definition of $\alpha(f)$ we see that $\alpha(f)(x) = y$ iff $y \in T(f(a))$ and $y = vf(a)i$. This can be recognised by a finite automaton since f is automatic. We leave the other cases and the rest of the proof to the reader.

Part 3) follows from Part 1) and Part 4) from Part 2).

The sizes of the automata that recognise the languages $T(A)$ and $T(R_i)$ are proportional to the sizes of the automata recognising A and R_i . To recognise the language $L(i)$ we need to recognise words on $L(i, \bar{a}, k)$ paths, use the automaton for R_i , and use the automaton that tells us if any given b is equal to the k th coordinate of \bar{a} . The size of the automaton that recognises $L(i)$ is thus proportional to the sizes of the automata presenting \mathcal{A} . Similarly, the size of the automaton recognising $E(\mathcal{A})$ is linear in the size of the presentation of \mathcal{A} . \square

Note: Results of this section can be obtained from the fact that all automatic structures are interpretable in N_p [4]. We, however, provided a direct method of transforming structures into graphs rather than doing this indirectly by using interpretations in N_p .

4. On Automatic Equivalence Relations

In this section we study automatic equivalence structures. We provide several methods of constructing automatic equivalence structures with different types of isomorphism invariants. An **equivalence structure** \mathcal{E} is (E, ρ) where ρ is an equivalence relation on domain E . For \mathcal{E} define the following two isomorphism invariants: $I_1(\mathcal{E}) =$

$\{n \leq \omega \mid \text{there is an equivalence class of size } n\}$, and $I_2(\mathcal{E}) = \{(n, m) \mid \text{there are exactly } m \text{ equivalence classes of size } n\}$. Clearly, $I_2(\mathcal{E})$ is a full isomorphism invariant in the sense that \mathcal{E} and \mathcal{E}' are isomorphic iff $I_2(\mathcal{E}) = I_2(\mathcal{E}')$. Also, $I_1(\mathcal{E})$ can be expressed in terms of $I_2(\mathcal{E})$. Our goal is to understand how these invariants behave in case \mathcal{E} is automatic. In [12] and [4] it is shown that \mathcal{E} has an automatic presentation over a unary alphabet iff $I_1(\mathcal{E})$ is finite and there are finitely many infinite equivalence classes. The results of this section show that the situation in the general case is complex.

For an equivalence structure \mathcal{E} we define \mathcal{E}_ω and \mathcal{E}_f as the restriction of \mathcal{E} to all elements in infinite and finite equivalence classes, respectively.

Lemma 2 *If the equivalence structure \mathcal{E} is automatic then so are \mathcal{E}_f and \mathcal{E}_ω . Moreover, \mathcal{E} has an automatic presentation iff \mathcal{E}_f does.*

Proof. Follows from the fact that \mathcal{E}_f is definable by a $FO(\exists^\infty)$ -formula and that \mathcal{E}_ω always has an automatic presentation. \square

Thus, in characterising automatic equivalence structures \mathcal{E} , we can assume that each equivalence class is finite. Therefore, from now on we always assume that $I_1(\mathcal{E})$ does not contain ω , and if $(n, m) \in I_2(\mathcal{E})$ then n is finite.

Lemma 3 *If \mathcal{E} is an automatic equivalence structure then it has an automatic presentation (E', ρ') satisfying the property that if $(x, y) \in \rho'$ then $|x| = |y|$.*

Proof. Suppose \mathcal{E} is automatic over Σ . Consider an automatic linear order \leq on E of the type ω so that if $x \leq y$ then $|x| \leq |y|$. We remark that one can always extend an automatic structure by such an order. The set $\{x \mid x \text{ is the } \leq\text{-longest element in its equivalence class}\}$ is regular. Define a new domain E' over alphabet $((\Sigma \cup \{1\})^*)^2$ as the set of pairs $(x, 1^n)$ where x is in the domain of \mathcal{E} and n is the length of the \leq -longest word in the ρ -equivalence class containing x . Note that E' is FA-recognisable. Define a new equivalence relation ρ' containing pairs $((x, 1^n), (y, 1^m))$ iff $(x, y) \in \rho$ and $n = m$. Then (E', ρ') is an automatic equivalence relation isomorphic to \mathcal{E} . \square

Corollary 1 *(also see [4]) Let \mathcal{E} be an infinite automatic equivalence relation where $|\Sigma| \geq 2$, and n_i be an increasing enumeration of the sizes of its equivalence classes. Then $n_i \leq 2^{O(i)}$. \square*

Next we build equivalence structures from languages $L \subset \Sigma^*$. Define an equivalence structure $\mathcal{E}(L) = (E, \sim_L)$ with $E = L$. Two strings x and y are \sim_L -equivalent if $|x| = |y|$ and $x, y \in L$. Here is an easy lemma:

Lemma 4 If L is regular then $\mathcal{E}(L)$ is an automatic structure. \square

One would like to characterise automatic equivalence structures in terms of $I_1(\mathcal{E})$ and $I_2(\mathcal{E})$. The next series of results provide several examples and standard constructions for building automatic equivalence structures whose isomorphism invariant $I_1(\mathcal{E})$ exhibits nontrivial behaviour.

Let L be a language over Σ . The growth of L is the function g_L defined as $g_L(n) = |\Sigma^n \cap L|$ for $n < \omega$. The following is implicit in [14].

Lemma 5 For any polynomial function p whose coefficients are positive integers there is a regular language L_p whose growth function is p .

Proof. Note that if L_1 and L_2 have growth rates p_1 and p_2 , respectively, and $L_1 \cap L_2 = \emptyset$ then their union has growth rate $p_1 + p_2$. So it is sufficient to exhibit for each $k \in \mathbb{N}$ a language L_{n^k} with growth rate n^k .

For $w \in \Sigma^*$, write w^+ for ww^* . Note that $A_k = 0^+1^+\dots k^+$ has growth $\binom{n-1}{k}$. Consider the languages $B_k = 0^+1^+\dots(k-1)^+k^*$. Then $B_k = A_{k-1} \cup A_k$. Hence the growth of B_k is $\binom{n-1}{k} + \binom{n-1}{k-1} = \binom{n}{k}$. Consider the languages C_k defined as the disjoint union of $k!$ copies of B_k . Then C_k has growth $n(n-1)\dots(n-k+1)$ which we write as n^k . We now make use of the standard identity $x^k = \sum_{i=0}^k S(k, i)x^i$ where the $S(k, i)$ are Stirling numbers of the first kind; that is the number of ways of partitioning a set of size k into i non-empty subsets. So $L_{n^k} = \bigcup_{i=0}^k \bigcup_{S(k,i)} C_i$, where the unions are taken to be disjoint, has the required growth. \square

Lemma 6 For any exponential function $e(n)$ of the form k^{an+b} , where $2 \leq k$ and a, b are positive integers, there exists a regular language whose growth function is exactly e .

Proof. Let $\Sigma = \{1, 2, \dots, k^a\}$. Then $L = \Sigma^*$ has growth k^{an} . The disjoint union of k^b many copies of L has growth k^{an+b} . \square

We note that for any regular language L its growth level is either bounded by a polynomial or is bounded by an exponential (see [1]).

Theorem 4 For any function f which is either a polynomial p whose coefficients are positive integers or exponential function k^{an+b} , where $k \geq 2$ and a, b are fixed positive integers, there exists an automatic equivalence relation \mathcal{E} such that $I_1(\mathcal{E}) = \{f(n) \mid n \geq 1\}$ and $I_2(\mathcal{E}) = \{(f(n), c) \mid n \geq 1\}$, with $c \leq \omega$ being a constant.

Proof. From Lemma 5 and Lemma 6 there exists a regular language L whose growth function is identical to f . By

Lemma 4 the automatic equivalence structure $\mathcal{E}(L)$ is the desired one. The theorem for case $c = 1$ is proved. Now note that for $c \leq \omega$, the c -fold disjoint union of automatic equivalence structures is automatic. \square

The next result shows that the second invariant $I_2(\mathcal{E})$ of automatic equivalence structures can also exhibit a complex behaviour. The invariant $I_2(\mathcal{E})$ defines the **height function** $h_{\mathcal{E}}$ as follows: $h_{\mathcal{E}}(n) = m$ if and only if $(n, m) \in I_2(\mathcal{E})$. Consider two functions f, g with domains \mathbb{N} . Their **Dirichlet convolution** is $(f * g)(n) = \sum_{ab=n} f(a)g(b)$, and their **Cauchy product** is $(f \# g)(n) = \sum_{a+b=n} f(a)g(b)$.

Proposition 1 Let \mathcal{H} be the class of height functions of automatic equivalence structures. Then \mathcal{H} is closed under addition, Dirichlet convolution and Cauchy product.

Proof. Let $\mathcal{E}_i = (E_i, \rho_i)$ for $i = 1, 2$ be two automatic equivalence structures with height functions f and g respectively. Without loss of generality, assume that the domains E_1 and E_2 are disjoint. Define the automatic equivalence structure $\mathcal{E}_{E_1 \cup E_2}$ as their disjoint union; that is, we assume that E_1 and E_2 are disjoint, and define the domain as $E_1 \cup E_2$ and the relation as $\rho = \rho_1 \cup \rho_2$. Then the height function of $\mathcal{E}_{E_1 \cup E_2}$ is $f + g$.

For the Dirichlet convolution define the direct product $\mathcal{E}_1 \times \mathcal{E}_2$ as the equivalence structure with domain $E_1 \times E_2$. Define two pairs (x_1, y_1) and (x_2, y_2) to be related if $(x_1, x_2) \in \rho_1$ and $(y_1, y_2) \in \rho_2$. Then this equivalence structure is automatic and has height $f * g$.

For the Cauchy product, let $T_i \subset E_i$ be the unary predicate that picks out the lexicographically least element from each equivalence class of \mathcal{E}_i . Define an equivalence structure with domain $(T_1 \times E_2) \cup (T_2 \times E_1)$. Define two pairs (x_1, y_1) and (x_2, y_2) to be related if either $[(y_1, y_2) \in \rho_{E_1 \cup E_2} \text{ and } x_1 = x_2]$ or $\{(x_1, y_2), (x_2, y_1)\} \subset \rho_{E_1 \cup E_2}$. Then this equivalence structure is automatic and has height $f \# g$. \square

Thus, for instance there is an automatic equivalence structure \mathcal{E} so that $I_1(\mathcal{E}) = \omega \setminus \{0\}$ and $h_{\mathcal{E}}(n)$ is the number of all pairs (i, j) such that $i \cdot j = n$.

We give an automata-theoretic characterisation of automatic equivalence structures (with finite equivalence classes).

Definition 5 An automatic binary relation R is a **regular enumeration** of a family \mathcal{F} of regular sets if $\mathcal{F} = \{R_x \mid x \in \text{dom}(R)\}$, where R_x is the projection $\{u \mid (x, u) \in R\}$.

We think of R as a mapping from $\text{dom}(R)$ onto \mathcal{F} . If R is a regular enumeration then one can always construct a regular one to one enumeration of \mathcal{F} since the relation $\{(x, y) \mid R_x = R_y\}$ is FA-recognisable.

Let R be a one to one regular enumeration of \mathcal{F} such that $R_x \cap R_y = \emptyset$ for $x \neq y$. Consider the structure $\mathcal{E}(R)$ with domain $\bigcup_{x \in \text{dom}(R)} R_x$ and binary relation $\{(u, v) \mid \exists x \in \text{dom}(R) : u, v \in R_x \& |u| = |v|\}$. Then $\mathcal{E}(R)$ is an equivalence structure. The proof of the following is immediate.

Proposition 2 *An equivalence structure is automatic if and only if it is of the form $\mathcal{E}(R)$ for a regular enumeration R .*

Example 1 *Let X and Y be nonempty regular languages such that no two words in Y are prefixes of each other. Consider the family $\mathcal{F} = \{yX \mid y \in Y\}$. The mapping $R : y \rightarrow yX$ is a one to one and regular enumeration of \mathcal{F} . Hence $\mathcal{E}(R)$ is automatic.*

The construction of $\mathcal{E}(R)$ is as general as possible because one can reverse the construction as follows. Let $\mathcal{E} = (E, \rho)$ be an automatic equivalence structure. We may assume that $(u, v) \in \rho$ implies that $|u| = |v|$. Form the set W of all the minimal elements (with respect to an automatic order \leq of type ω on the set of all words) from each equivalence class. Consider $R = \{(w, v) \mid w \in W, v \in V, (w, v) \in \rho\}$. Clearly, R is a regular one to one enumeration of the ρ -equivalence classes and $\mathcal{E}(R)$ is isomorphic to \mathcal{E} .

5. On Automatic Linearly Ordered Sets

Here we explain how to convert automatic equivalence structures \mathcal{E} into certain types of linearly ordered (lo) sets $\mathcal{L}_{\mathcal{E}}$ so that $\mathcal{L}_{\mathcal{E}}$ and \mathcal{E} can be recovered from each other. This will show that the study of automatic lo sets is at least as complex as automatic equivalence structures.

Let $\mathcal{L} = (L, \leq)$ be a lo set. For $x, y \in L$ define the **interval** $[x, y] = \{z \mid x \leq z \leq y\}$ if $x \leq y$ and $[x, y] = \{z \mid y \leq z \leq x\}$ if $y < x$. We say that the elements $x, y \in L$ are **in the same block** if $[x, y]$ is finite, and we write B for this equivalence relation on L . Having the relation B , define a new lo set \mathcal{L}_B by factorising L by B as follows. The elements of \mathcal{L}_B are the equivalence classes; and $x_B \leq_B y_B$ if $x \leq y$, where x_B is the equivalence class containing x .

Lemma 7 *If \mathcal{L} is an automatic lo set then the block relation B is FA recognisable. Hence the factor \mathcal{L}_B is also automatic. \square*

The idea of factorisation suggests relating automatic equivalence relations with lo sets. We need a definition.

Definition 6 *We denote Q the type of the lo set of rationals. We say that a lo set \mathcal{L} has Q -rank 1 if \mathcal{L}_B is isomorphic to either Q or $1 + Q$ or $Q + 1$, where $1 + Q$ ($Q + 1$) is the lo set of rationals with the least (greatest) element.*

Let \mathcal{L} be a lo set of Q -rank 1. Define the set $I(\mathcal{L}) = \{(n, m) \mid \mathcal{L} \text{ has } m \text{ blocks of size } n\}$. Write $U(x)$ for the unary relation on L stating that x is in some dense interval. Define $\mathcal{E}(\mathcal{L})$ as the equivalence structure with domain L and relation $B \cup (U \times U)$.

Theorem 5 *From any automatic equivalence structure \mathcal{E} it is possible to construct an automatic lo set $\mathcal{L}_{\mathcal{E}}$ of Q -rank 1 so that*

$$\begin{aligned} I_2(\mathcal{E}(\mathcal{L}_{\mathcal{E}})) &= \{(n, m) \mid (n, m) \in I_2(\mathcal{E}), n < \omega\} \\ &\cup \{(\omega, m+1) \mid (\omega, m) \in I_2(\mathcal{E})\}. \end{aligned}$$

Proof. Suppose that $\mathcal{E} = (E, \rho)$ has an automatic presentation. Let \prec be an automatic well ordering on the set E . Write \prec_A for \prec restricted to domain A . Order the equivalence classes of \mathcal{E} by \prec as follows. We write $x_{\rho} \prec_E y_{\rho}$ iff the \prec -minimal element in x_{ρ} is \prec the \prec -minimal element in y_{ρ} . List the equivalence classes of \mathcal{E} as $\{B_i\}$ for $i < \omega$ where $i \leq j$ iff $B_i \prec_E B_j$. The required lo set $\mathcal{L}_{\mathcal{E}}$ is then $\Sigma_i(B_i + D)$, where $B_i = (B_i, \prec_{B_i})$ and D is an automatic lo of type Q over a new alphabet. The lo set \mathcal{L} has an automatic presentation. To this end we note that if x_i is the \prec -minimal element of B_i , then we order the set $x_i D = \{x_i d \mid d \in D\}$ as $x_i d_1$ is less than $x_i d_2$ iff $d_1 \leq_D d_2$. \square

Corollary 2 *For any function g which is either a polynomial p whose coefficients are positive integers or exponential function k^{an+b} , where $k \geq 2$ and a, b are fixed positive integers, there exists an automatic lo set \mathcal{L} of Q -rank 1 such that $I(\mathcal{L}) = \{(g(n), 1) \mid n < \omega\} \cup \{(\omega, 1)\}$. \square*

6. On Automatic Permutation Structures

A permutation structure \mathcal{A} is (A, f) where f is a bijection on A . For an $a \in A$, the set $\{f^i(a) \mid i < \omega\}$ is an **orbit** of f . As for the equivalence structures, define two isomorphism invariants $I_1(\mathcal{A}) = \{n \leq \omega \mid \text{there is an orbit of size } n\}$, and $I_2(\mathcal{A}) = \{(n, m) \mid \text{there are exactly } m \text{ orbits of size } n\}$. Then I_2 is a full isomorphism invariant. In [13] and in [4] it is shown that \mathcal{A} has an automatic presentation over a unary alphabet if and only if $I_1(\mathcal{A})$ is finite and there are finitely many infinite orbits.

Any automatic equivalence structure \mathcal{E} over Σ^* can be turned into an automatic permutation structure $\mathcal{A}(\mathcal{E})$ as follows. Let \leq be an automatic well order of type ω on Σ^* . For each $x \in E$ we proceed as follows. If x is in \mathcal{E}_f and is not the maximal element in its equivalence class then $f(x)$ is the minimal y ρ -equivalent to x . Otherwise, $f(x)$ is the minimal element in the equivalence class containing x . If $x \in \mathcal{E}_{\omega}$ then f transforms the equivalence class into \mathbb{Z} -type chain, namely the structure isomorphic to (\mathbb{Z}, S)

where S is the successor function on the integers. Note that $I_2(\mathcal{E}) = I_2(\mathcal{A}(\mathcal{E}))$. Hence, the result similar to Theorem 4 holds true for automatic permutation structures.

We want to show that the isomorphism invariants $I_1(\mathcal{A})$ of automatic permutation structures can be related to the running times of Turing machines (TMs). Let T be a TM over input alphabet Σ . Its configuration graph $C(T)$ consists of the set of all configurations of T , with an edge from c to d if T can move from c to d in a single transition. Here is a simple lemma:

Lemma 8 *For any TM T the configuration graph $C(T)$ is automatic. Further, the set of all vertices with with outdegree (indegree) 0 is FA-recognisable. \square*

Definition 7 *A TM T is reversible if every vertex in $C(T)$ has indegree and outdegree at most one.*

Bennett [2] showed that any deterministic TM T can be simulated by a reversible TM R . Furthermore, running times of these machines differ by a constant factor. For the sake of completeness, we sketch the proof.

A transition of T is a quintuple $(\sigma, q, \delta, d, s) \in \Delta$ where $\sigma, \delta \in \Sigma$, $q, s \in Q$ and $d \in \{L, R\}$. On input w , R runs as T would, but also saves each of T 's transitions on a separate ‘history’ tape. Once the simulated T has halted, R copies the output to another tape. It then retraces the steps that T took, in reverse, deleting the saved transitions one at a time, resulting in R having the original input w printed on one tape, a blank ‘history’ tape, and the output $T(w)$ on the third tape. This three tape TM R is itself simulated by a single tape machine. So, the reason that R is reversible is that if a configuration c of T has indegree greater than 1, then the transitions corresponding to each edge into c are distinct. Since the corresponding configuration of R codes these transitions, the particular configuration of T which preceded c is uniquely determined.

Let $Time_T(w)$ be the number of steps T takes to halt on w , so that $Time_T(w) = \omega$ if T does not halt on w .

Theorem 6 *For every reversible TM T , there corresponds an automatic permutation structure $\mathcal{A}(T)$ for which $I_1(\mathcal{A}(T)) = \{Time_T(w) \mid w \in \Sigma^*\}$.*

Proof. Let the configuration graph of T be $C(T) = (C, E)$. Assume that the unique initial state of T is not a final state. Then $Time(w) > 0$ for all w . Let $I, O \subset C$ respectively be the set of configurations with indegree 0 and outdegree 0. For each $c \in C$, let \tilde{c} be a symbol not in C . Let $\tilde{C} = \{\tilde{c} \mid c \in C\}$. Define a permutation function f on domain $C \cup \tilde{C}$ as follows. If $(c, d) \in E$ then $f(c) = d$ and $f(\tilde{d}) = \tilde{c}$; If $c \in I$ and $(c, d) \in E$ then $f(\tilde{d}) = \tilde{c}$; If $d \in O$ and $(c, d) \in E$ then $f(\tilde{d}) = \tilde{c}$.

Consider the structure $(C \cup \tilde{C}, E, I, O, \sim)$ where \sim is the function with domain C mapping c to \tilde{c} . It is automatic over alphabet $\Sigma \cup \tilde{\Sigma}$. We conclude that the structure $(C \cup \tilde{C}, f)$ is also automatic. Factor this structure by the equivalence relation satisfying pairs of the form (c, \tilde{c}) and (\tilde{c}, c) for $c \in I \cup O$. Write (D, g) for the resulting automatic permutation structure. If $Time_T(w) = n$ then the (D, g) has an orbit of length $2n$. So, define the desired structure $\mathcal{A}(T) = (D, h)$ where $h = g \circ g$. \square

Theorem 7 *It is undecidable whether two automatic permutation structures are isomorphic.*

Proof. For a deterministic TM T' , construct an equivalent reversible TM T and the structure $\mathcal{A}(T)$. T halts on no word iff \mathcal{A} is isomorphic to the permutation structure with only infinitely many infinite chains of type \mathbb{Z} . \square

Blumensath [3] also proved undecidability of the isomorphism problem for automatic structures by an implicit construction of reversible TMs.

7. Conclusion

We would like to have a characterisation of natural isomorphism invariants of automatic structures. Ideally, we would like these characterisations to give us some useful information about the complexity-theoretic nature of the structures from logical and algebraic points of view. When the structures are automatic over a unary alphabet, characterisation for some common structures are known [4],[12]. These characterisations imply that theories of these structures are computationally accessible and show the algebraic nature of the structures. In this paper our aim was to show difficulties involved in the non-unary case. Theorem 3 reduces the study of automatic structures to automatic graphs. Theorem 4 and Theorem 5 are initial steps in understanding the isomorphism invariants of some simple structures. Finally, Theorem 6 exhibits a nontrivial relationship between running times of TMs and automatic structures. The paper shows that more work remains to be done in understanding automatic structures and their complexities. We deal with some of them in upcoming papers.

We thank the referees for their general comments. In particular, one referee suggested including the Cauchy product in Proposition 1.

References

- [1] M. Benedikt, L. Libkin, T. Schwentick, and L. Segoufin. A Model-Theoretic approach to regular string relations. In *Proceedings of the 16th Annual IEEE Symposium on Logic in Computer Science (LICS'01)*, pages 431–440, June 16–19 2001.

- [2] C. H. Bennett. Logical reversibility of computation. *IBM Journal of Research and Development*, 17(6):525–532, Nov. 1973.
- [3] A. Blumensath. personal communication.
- [4] A. Blumensath. *Automatic Structures*. Diploma thesis, RWTH Aachen, 1999.
- [5] A. Blumensath and E. Grädel. Automatic structures. In *15th Symposium on Logic in Computer Science (LICS' 00)*, pages 51–62, June 2000.
- [6] C. Delhomme, V.Goranko, and T.Knapik. Automatic linear orderings. personal communication.
- [7] S. Eilenberg, C. Elgot, and J. Shepherdson. Sets recognised by n -tape automata. *Journal of Algebra*, 13(4):447–464, 1969.
- [8] D. B. H. Epstein, J. W. Cannon, D. F. Holt, S. V. F. Levy, M. S. Paterson, and W. P. Thurston. *Word processing in groups*. Jones and Bartlett, 1992.
- [9] E. Grädel. Simple interpretations among complicated theories. *Information Processing Letters*, 35(5):235–238, 1990.
- [10] W. A. Hodges . *Model Theory*. Cambridge University Press, Cambridge, 1993.
- [11] B. Khoussainov and A. Nerode. Automatic presentations of structures. *Lecture Notes in Computer Science*, 960:367–392, 1995.
- [12] B. Khoussainov and S. Rubin. Graphs with automatic presentations over a unary alphabet. *Journal of Automata, Languages and Combinatorics*, 6(4), 2001.
- [13] S. Rubin. Finite automata and well ordered sets. *New Zealand Journal of Computing*, 7(2):39–46, 1999.
- [14] A. Szilard, S. Yu, K. Zhang, and J. Shallit. Characterizing regular languages with polynomial densities. In I. Havel and V. Koubek, editors, *Mathematical Foundations of Computer Science 1992, 17th International Symposium*, volume 629 of *lncs*, pages 494–503, 24–28 Aug. 1992.

On Automatic Partial Orders

Bakhadyr Khoussainov

Department of Computer Science

University of Auckland, New Zealand

bmk@cs.auckland.ac.nz

Sasha Rubin

Department of Mathematics

University of Auckland, New Zealand

rubin@math.auckland.ac.nz

Frank Stephan

Mathematical Institute

University of Heidelberg, Germany

fstephan@math.uni-heidelberg.de

Abstract

We investigate partial orders that are computable, in a precise sense, by finite automata. Our emphasis is on trees and linear orders. We study the relationship between automatic linear orders and trees in terms of rank functions that are versions of Cantor-Bendixson rank. We prove that automatic linear orders and automatic trees have finite rank. As an application we provide a procedure for deciding the isomorphism problem for automatic ordinals. We also investigate the complexity and definability of infinite paths in automatic trees. In particular, we show that every infinite path in an automatic tree with countably many infinite paths is a regular language.

1. Introduction

Consider a class of infinite structures, such as the class of graphs, partial orders, trees, groups, or lattices, etc. A given structure in this class may or may not be computable. If it is one then naturally asks whether or not the structure, or algorithmic problems of the structure, are feasibly computable. The area of automatic structures studies those structures that are computable, in a certain precise sense, by finite automata. The automata in this paper are synchronous automata operating on finite words. Using the closure of these automata under boolean operations and projection, we get that the first order theory of an automatic structure is decidable (see, e.g. [10]). This is a result that motivates study of automatic structures in computer science. For example, a related concept to the one presented here is that of automatic groups from computational group theory [7]. There they prove that a finitely generated automatic group is finitely presentable and that its word problem is

solvable in quadratic time. The general notion of structures presentable by automata has been recently studied in [1],[2],[5],[8],[10],[12]. Throughout this paper we will use the following more general theorem proved by Grädel and Blumensath (see [2]) without explicit mention.

Theorem 1.1 *Given an automatic structure \mathcal{A} and a relation R which is first order definable (with the quantifier \exists^∞ which stands for “there exist infinitely many”) in \mathcal{A} , one can effectively construct an automaton recognising R .*

This paper studies automatic partial orderings with an emphasis on trees and linear orderings. A **partial order** (**partial ordering**) is a pair (A, \preceq) such that \preceq is a reflexive, transitive and anti-symmetric binary relation on the nonempty domain A . A **linear order** \mathcal{L} is a partial order (L, \leq) in which \leq is total, that is $\forall x \forall y (x \leq y \vee y \leq x)$. A general problem of our interest concerns characterising the isomorphism types of the automatic linear orders. Classically, linear orderings are characterised in terms of scattered and dense linear orderings as follows. We say that \mathcal{L} is **dense** if for all distinct a and b in L with $a < b$ there exists an $x \in L$ with $a < x < b$. There are only five types of countable dense linear orderings up to isomorphism: the order of rational numbers with or without least and greatest elements, and the order type of the trivial linear order with exactly one element. We say that \mathcal{L} is **scattered** if it does not contain a nontrivial dense subordering. Examples, of scattered linear orders are finite sums of cartesian products of order types of ω and Z (integers).

Theorem 1.2 see [14, Theorem 4.9] *Every countable linear ordering \mathcal{L} can be represented as a dense sum of countable scattered linear orderings.*

The scattered linear orderings can be characterised inductively, whereby to each linear order \mathcal{L} one associates a

countable ordinal – called the *FC*–rank of \mathcal{L} , a version of Cantor-Bendixson rank for topological spaces. One of our results in this paper gives an upper bound on the *FC*–rank of automatic linear orders. For scattered linear orders, the *FC*–rank coincides with the *VD*–rank (these are defined in the next section).

Theorem 3.5 *If \mathcal{L} is an automatic linear order, then its *FC*–rank is finite.*

The proof of this theorem generalises a novel technique of Delhomme who gives a full characterisation of automatic ordinals.

Corollary 1.3 [4] *An ordinal α is automatic if and only if $\alpha < \omega^\omega$.*

Another class of structures discussed in this paper are trees. A **tree** $\mathcal{T} = (T, \preceq)$ is a partial order that has a minimum element and in which every set of the form $\{y \mid y \prec x\}$ forms a finite linear order. Elements of trees are called nodes. A node $y \in T$ is an immediate successor of $x \in T$ if $x \prec y$ and there does not exist z for which $x \prec z \prec y$. A tree \mathcal{T} is **finitely branching** if each node $x \in T$ has only finitely many immediate successors. A **path** of a tree (T, \preceq) is a subset $P \subseteq T$ which is linearly ordered, closed downward (that is, whenever $y \in P$ and $x \preceq y$ then $x \in P$) and maximal (with respect to set theoretic inclusion) with these properties. An **infinite path** is a path P where $|P|$ is infinite. We are interested in understanding algebraic, model-theoretic as well as computational properties of automatic trees.

In analogy to Kleene-Brouwer orderings in which trees are associated with linear orderings (see [13]), we build linear orderings from trees. This transformation preserves automaticity, and associates the Cantor-Bendixson rank (*CB*–rank for short) of trees with the *VD*–ranks of linear orders obtained. Informally, the *CB*–rank of the tree tells us how big the tree is in terms of ordinals (see [9] for example). This relationship between trees and linear orders gives us the next result:

Theorem 5.5 *The *CB*–rank of an automatic tree with countably many infinite paths is finite.*

It is known that every infinite finitely branching tree has an infinite path (König's Lemma). The proof of this fact does not produce an infinite path constructively. In fact, there are even examples of computable finitely branching trees with *exactly* one infinite path, and the path is *not* computable. Moreover, if one omits the assumption that the tree is finitely branching then there are examples of computable trees in which every infinite path is not even arithmetical (see [13]). This negative phenomenon fails dramatically when one considers automatic trees, and not only finitely branching ones.

Theorem 4.1 *If an automatic tree has an infinite path, then it has a regular infinite path.*

We can significantly strengthen this theorem under the assumption that the tree has at most countably many paths. Indeed, from Theorem 5.4 one can derive that if an automatic finitely branching tree \mathcal{T} has countably many infinite paths then every path of \mathcal{T} is regular. This is because the set of paths in such trees is definable. However, we can even omit the assumption that the tree is finitely branching:

Theorem 4.4 *If an automatic tree has countably many infinite paths then every infinite path in it is regular.*

All classical definitions and unproved results on linear orderings can be found in Rosenstein [14]. Countable means finite or countably infinite. All structures are assumed to be countable. Definable means first order definable with the additional quantifier \exists^∞ .

2. Preliminaries

A thorough introduction to automatic structures can be found in [1] and [10]. A recent survey paper [11] discusses the basic results and possible directions for future work in the area. Familiarity with the basics of finite automata theory is assumed though for completeness the necessary definitions are included here.

A *finite automaton* \mathcal{A} over an alphabet Σ is a tuple (S, ι, Δ, F) , where S is a finite set of *states*, $\iota \in S$ is the *initial state*, $\Delta \subset S \times \Sigma \times S$ is the *transition table* and $F \subset S$ is the set of *final states*. A *computation* of \mathcal{A} on a word $\sigma_1\sigma_2\dots\sigma_n$ ($\sigma_i \in \Sigma$) is a sequence of states, say q_0, q_1, \dots, q_n , such that $q_0 = \iota$ and $(q_i, \sigma_{i+1}, q_{i+1}) \in \Delta$ for all $i \in \{0, 1, \dots, n-1\}$. If $q_n \in F$, then the computation is *successful* and we say that automaton \mathcal{A} *accepts* the word. The *language* accepted by the automaton \mathcal{A} is the set of all words accepted by \mathcal{A} . In general, $D \subset \Sigma^*$ is *finite automaton recognisable*, or *regular*, if D is equal to the language accepted by \mathcal{A} for some finite automaton \mathcal{A} .

Classically finite automata recognise sets of words. The following definitions extends recognisability to relations of arity n , called *synchronous n –tape automata*. Informally a synchronous n –tape automaton can be thought of as a one-way Turing machine with n input tapes [6]. Each tape is regarded as semi-infinite having written on it a word in the alphabet Σ followed by an infinite succession of blanks, \diamond symbols. The automaton starts in the initial state, reads simultaneously the first symbol of each tape, changes state, reads simultaneously the second symbol of each tape, changes state, etc., until it reads a blank on each tape. The automaton then stops and accepts the n –tuple of words if it is in a final state. The set of all n –tuples accepted by the automaton is the relation recognised by the automaton. Here is a formalisation:

Definition 2.1 Let Σ_\diamond be $\Sigma \cup \{\diamond\}$ where $\diamond \notin \Sigma$. The convolution of a tuple $(w_1, \dots, w_n) \in (\Sigma^*)^n$ is the tuple $(w_1, \dots, w_n)^\diamond \in ((\Sigma_\diamond)^*)^n$ formed by concatenating the least number of blank symbols, \diamond , to the right ends of the w_i , $1 \leq i \leq n$, so that the resulting words have equal length. The convolution of a relation $R \subset (\Sigma^*)^n$ is the relation $R^\diamond \subset ((\Sigma_\diamond)^*)^n$ formed as the set of convolutions of all the tuples in R .

Definition 2.2 An n -tape automaton on Σ is a finite automaton over the alphabet $(\Sigma_\diamond)^n$. An n -ary relation $R \subset \Sigma^{*n}$ is finite automaton recognisable or regular if its convolution R^\diamond is recognisable by an n -tape automaton.

We now relate n -tape automata to structures. A *structure* \mathcal{A} consists of a set A called the *domain* and some constants, relations and operations on A . We may assume that \mathcal{A} only contains relational and constant predicates as the operations can be replaced with their graphs. We write $\mathcal{A} = (A, R_1^A, \dots, R_k^A, c_0^A, \dots, c_t^A)$ where R_i^A is an n_i -ary relation on \mathcal{A} and c_j^A is a constant element of \mathcal{A} .

Definition 2.3 A structure \mathcal{A} is automatic over Σ if its domain $A \subset \Sigma^*$ and the relations $R_i^A \subset \Sigma^{*n_i}$ all are finite automaton recognisable.

An isomorphism from a structure \mathcal{B} to an automatic structure \mathcal{A} is an automatic presentation of \mathcal{B} in which case \mathcal{B} is called automatically presentable (over Σ). A structure is called automatic if it is automatic over some alphabet.

Examples of automatic or automatically presentable structures are Presburger arithmetic $(\omega, S, +, 0)$, the group of integers $(\mathbb{Z}, +)$, the Boolean algebra of finite or co-finite subsets of ω , the word structure $(\{0, 1\}^*, L, R, E, \preceq)$, where for all strings $x, y \in \{0, 1\}^*$ we have $L(x) = x0$, $R(x) = x1$, $E(x, y)$ iff $|x| = |y|$, and \preceq is the lexicographical order. Examples of automatic or automatically presentable linear orders are (ω, \leq) , (\mathbb{Z}, \leq) and the order on rationals (\mathbb{Q}, \leq) . Moreover, if $\mathcal{L}_1 = (L_1, \leq_1)$ and $\mathcal{L}_2 = (L_2, \leq_2)$ are automatic linear orders then so are their sum and product. Below we present two examples that generate automatic trees.

Example 2.4 Let R be a regular language. Consider the partial order $\mathcal{T} = (\text{Pref}(R), \leq)$, where $\text{Pref}(R)$ is the language of all prefixes of strings from R , and \leq is the prefix relation. Then \mathcal{T} is an automatic tree.

Example 2.5 Let R be a regular language. Consider the partial order $\mathcal{T} = (R, \leq)$, where $x \leq y$ iff $x = y$ or $|x| < |y|$ and x is lexicographically smallest among all $x' \in R$ such that $|x| = |x'|$. Then \mathcal{T} is an automatic tree.

We now need some facts and notations about linear orders. Write ω for the type of the positive integers, ω^* for the negative integers, ζ for the integers, η for the rationals and \mathbf{n}

for the finite order on n elements. A **closed interval** written $[x, y]$ is $\{z \mid x \leq z \leq y\}$ if $x \leq y$ and $\{z \mid y \leq z \leq x\}$ otherwise. If \mathcal{L} is a linear ordering, then unless specified we denote its domain by L and ordering by \leq_L or simply \leq .

Definition 2.6 Consider a linear order I as an index set for a set of linear orderings $\{\mathcal{A}_i\}_{i \in I}$. The **I -sum**

$$\mathcal{L} = \Sigma\{\mathcal{A}_i \mid i \in I\}$$

is the linear order with domain $\cup_i A_i$. For $x \in A_i, y \in A_j$ define $x \leq_L y$ if $(i <_I j) \vee (i = j \wedge x \leq_{A_i} y)$.

We refer to the case when I is dense as a **dense sum**. We need the following definition of VD -rank and the class VD (very discrete) of scattered linear orders.

Definition 2.7 For each countable ordinal α , define the set VD_α of linear orders inductively as

1. $VD_0 := \{\mathbf{0}, \mathbf{1}\}$, where $\mathbf{0}$ is the empty ordering and $\mathbf{1}$ is the ordering with exactly one element.
2. $VD_\alpha :=$ the set of linear orderings formed as I -sums where the $\{\mathcal{A}_i\}$ are linear orderings from $\bigcup\{VD_\beta \mid \beta < \alpha\}$ and I is of the type ω, ω^*, ζ or \mathbf{n} for some $n < \omega$.

Define the class VD as the union of the VD_α 's. The **VD -rank** of a linear ordering $\mathcal{L} \in VD$, written $VD(\mathcal{L})$, is the least ordinal α such that $\mathcal{L} \in VD_\alpha$.

Example 2.8 Let $\mathcal{L}_1 = \Sigma\{\zeta + \mathbf{n} \mid n \in \omega\}$, $\mathcal{L}_2 = (\zeta \cdot \zeta) \cdot \zeta$. Then $VD(\mathcal{L}_1) = 2$, $VD(\mathcal{L}_2) = 3$ and $VD(\mathcal{L}_1 + \mathcal{L}_2) = 4$. In general, if $n = \max(VD(\mathcal{L}_1), VD(\mathcal{L}_2))$, then $n \leq VD(\mathcal{L}_1 + \mathcal{L}_2) \leq n + 1$.

Example 2.9 Let α, β be countable ordinals. Then $VD(\beta) \leq \alpha$ iff $\beta \leq \omega^\alpha$. In particular, $VD(\omega^\alpha) = \alpha$.

Theorem 2.10 [14, Theorem 5.24] A countable linear ordering \mathcal{L} is scattered if and only if \mathcal{L} is in VD .

There is an alternative definition of ranking that we use in some of the proofs. We proceed with the definitions.

Definition 2.11 A **condensation map** is a mapping c from L to non-empty intervals of L such that $c(y) = c(x)$ whenever $y \in c(x)$. The **condensation** of \mathcal{L} is the linear order $c[\mathcal{L}]$ whose domain consists of the collection of non-empty intervals $c(x)$ for $x \in L$ ordered by $c(x) \ll c(y)$ if $\forall x_1 \in c(x) \forall y_1 \in c(y) (x_1 < y_1)$.

Define the **iterated condensation** c^α as a mapping from \mathcal{L} to a set of non-empty intervals of \mathcal{L} inductively as

1. $c^0(x) = \{x\}$ for all $x \in L$.

2. $c^{\beta+1}(x) = \{y \in L \mid c(c^\beta(x)) = c(c^\beta(y))\}$,
3. $c^\lambda(x) = \bigcup\{c^\beta(x) \mid \beta < \lambda\}$ for limit ordinal λ .

Example 2.12 As an illustration, we prove that every countable linear ordering can be represented as a dense sum of scattered linear orderings (Theorem 1.2).

Proof The mapping $\{y \mid [x, y] \text{ is scattered}\}$, written $c_S(x)$, is a condensation since if $a \in c_S(x)$ then for all y , $[a, y]$ does not contain a dense subordering if and only if $[x, y]$ does not contain a dense subordering. Then $c_S[\mathcal{L}]$ is dense since for $c_S(x) \ll c_S(y)$, if there is no z with $c_S(x) \ll c_S(z) \ll c_S(y)$ then $[x, y]$ is scattered, contrary to assumption. Hence $c_S[\mathcal{L}]$ is a dense linear ordering and $\mathcal{L} = \sum\{a \mid a \in c[\mathcal{L}]\}$. Finally, note that each $a = c_S(x) \in c[\mathcal{L}]$ is scattered. \square

An important condensation is $\{y \mid [x, y] \text{ is finite}\}$, written $c_{FC}(x)$ to which ones refers as a finite condensation. The idea here is that $c_{FC}^1(x)$ is the set of elements of \mathcal{L} that are only finitely far away from x ; $c_{FC}^2(x)$ is the set of elements of \mathcal{L} that are in intervals of $c_{FC}[\mathcal{L}]$ which themselves are only finitely far away in $c_{FC}[\mathcal{L}]$ from the interval $c_{FC}^1(x)$. The least ordinal α such that $c_{FC}^\beta(x) = c_{FC}^\alpha(x)$ for all $x \in \mathcal{L}$ and $\beta \geq \alpha$ is called the **FC-rank** of \mathcal{L} , written $FC(\mathcal{L})$. From now on, we write c for c_{FC} .

Example 2.13 A linear order \mathcal{L} is dense if and only if its FC-rank is 0. Moreover, \mathcal{L} is scattered if and only if $c^\alpha[\mathcal{L}] \simeq \mathbf{1}$ for some ordinal α .

The following theorem connects FC-ranks and VD -ranks of scattered linear orderings.

Theorem 2.14 [14, Theorem 5.24] If \mathcal{L} is scattered then its VD -rank equals its FC-rank.

If $A \subset L$ then we denote the condensation taking place within the set L by c and the condensation taking place relative to A by c_A . That is, c_A is just c with A replacing \mathcal{L} in the definition. In this case we will also write $c_A(x)$, $c_A[\mathcal{A}]$ and \ll_A . Here are some useful properties, where the third (non-cited) one can be proven inductively.

- Lemma 2.15**
1. [14, Lemma 5.14] If \mathcal{L} is scattered and $M \subset L$ then $FC(M) \leq FC(\mathcal{L})$.
 2. [14, Lemma 5.13 (2)] $FC(c^\alpha(x)) \leq \alpha$ for every α , $x \in L$, and $c^\alpha(x)$ is a scattered interval of \mathcal{L} for all α .
 3. For every $x, y \in L$, if $[x, y]$ is scattered then $c^\alpha(x) = c^\alpha(y)$ if and only if $FC([x, y]) \leq \alpha$.
 4. [14, Exercise 5.12 (1)] If I is an interval of \mathcal{L} then for every α , $c_I^\alpha(x) = c^\alpha(x) \cap I$.

3. Ranks of Automatic Linear Orderings

This section is devoted to the proof of Theorem 3.5. For this, we prove three propositions. Delhomme's idea [4] is to analyse the transition diagram of the automaton describing an ordinal, which we imitate in the proof of Theorem 3.4. As a matter of convenience, we introduce the following variation of rank.

Definition 3.1 If \mathcal{L} is scattered, define its VD_* -rank as being the least ordinal α such that \mathcal{L} can be written as a finite sum of orderings of VD -rank $\leq \alpha$.

For example, it is not hard to show that $VD(\omega) = VD_*(\omega) = 1$ and that $\omega 2 + 1$ has VD -rank 2 but VD_* -rank 1. We list two basic properties.

1. In general, $VD_*(\mathcal{L}) \leq VD(\mathcal{L}) \leq VD_*(\mathcal{L}) + 1$.
2. $c^\alpha[\mathcal{L}]$ is a finite linear order if and only if $VD_*(\mathcal{L}) \leq \alpha$.

Proposition 3.2 Let $\mathcal{L} = (L, \leq)$ be a scattered linear ordering. Consider a finite partition of the domain $L = A_1 \cup A_2 \cup \dots \cup A_k$. Then there exists some $1 \leq i \leq k$ with $VD_*(A_i) = VD_*(\mathcal{L})$.

Proof We prove the proposition for $k = 2$; the general case reduces to this case. Thus, assume that $A_0 \subset L$ and $A_1 = L \setminus A_0$. We need to show, by induction on $VD_*(\mathcal{L})$, that $VD_*(\mathcal{L}) = VD_*(A_\epsilon)$ for some $\epsilon \in \{0, 1\}$. The case when $VD_*(\mathcal{L}) = 0$ or $VD_*(\mathcal{L}) = 1$ is checked easily. Assume that the proposition is true for all \mathcal{L} such that $VD_*(\mathcal{L}) < \alpha$.

Suppose $VD_*(\mathcal{L}) = \alpha$. Then \mathcal{L} is a finite sum of orders of VD -rank at most α . In particular, at least one of these must have VD -rank exactly α . Call it \mathcal{M} . Then \mathcal{M} is an I -sum of linear orders $\{\mathcal{L}_i\}$ of VD -rank $< \alpha$, where I is of the type ω, ω^*, ζ or \mathbf{n} for some $n < \omega$. We may assume that \mathcal{M} is chosen so that I is not finite, for if every such \mathcal{M} were a finite sum of orders of VD -rank $< \alpha$, then \mathcal{L} would have VD_* -rank $< \alpha$. So assume that I is infinite, say of type ω (the other infinite cases are similar).

For the first case, suppose that $\alpha = \beta + 1$. We can assume that there are infinitely many i such that $VD_*(\mathcal{L}_i) = \beta$, for otherwise we could write \mathcal{M} as a finite sum of orders of VD -rank β , contrary to assumption. For each i let $A_{\epsilon,i} = L_i \cap A_\epsilon$, where $\epsilon \in \{0, 1\}$. Hence, applying the induction hypothesis to \mathcal{L}_i , we see that there is an $\epsilon \in \{0, 1\}$ and infinitely many j 's such that $VD_*(A_{\epsilon,j}) = VD_*(\mathcal{L}_j) = \beta$. Hence A_ϵ contains a subset which is an ω -sum of linear orders of VD_* -rank β . Therefore, $VD_*(A_\epsilon) > \beta$, and so $VD_*(A_\epsilon) = \alpha$ as required.

For the second case, suppose that α is a limit ordinal. So \mathcal{M} is an ω -sum of linear orders $\{\mathcal{L}_i\}$ such the VD -rank

of each \mathcal{L}_i is less than α , and the supremum of the VD -ranks of \mathcal{L}_i is α . Using the notation of the case above, and applying induction, we see that there is an $\epsilon \in \{0, 1\}$ and infinitely many j 's such that $VD_*(\mathcal{A}_{\epsilon,j}) = VD_*(\mathcal{L}_j)$, and the supremum of the VD_* -ranks of these $\mathcal{A}_{\epsilon,j}$'s is α . Then $VD_*(\mathcal{A}_\epsilon) = \alpha$ as required. \square

Proposition 3.3 *Let \mathcal{L} have FC-rank α . Then for every $\beta < \alpha$ there exists a closed scattered interval of \mathcal{L} of rank $\beta + 1$.*

Proof Fix $\beta < \alpha$. Since \mathcal{L} has FC-rank $> \beta$, by definition there is some $x \in L$ such that $c^\beta(x) \neq c^{\beta+1}(x)$. Pick $y \in c^{\beta+1}(x) \setminus c^\beta(x)$. Then $c^\beta(x) \neq c^\beta(y)$ and $c^{\beta+1}(x) = c^{\beta+1}(y)$. Recall that $c_{[x,y]}^\beta$ is the condensation mapping c^β within the interval $[x, y]$. Hence $c_{[x,y]}^\beta(x) \neq c_{[x,y]}^\beta(y)$ and $c_{[x,y]}^{\beta+1}(x) = c_{[x,y]}^{\beta+1}(y)$. The first fact implies that $VD([x, y]) > \beta$ and the second fact implies that $VD([x, y]) \leq \beta + 1$. Hence $VD([x, y]) = \beta + 1$ as required. \square

Now we prove the following theorem:

Theorem 3.4 *The VD -rank of every automatic scattered linear ordering is finite.*

Proof Suppose \mathcal{L} is automatic scattered linear over Σ^* . Let $(Q_\leq, \iota_\leq, \Delta_\leq, F_\leq)$ be the 2-tape automaton recognising the ordering of \mathcal{L} . Let $(Q_A, \iota_A, \Delta_A, F_A)$ be the 3-tape automaton recognising the definable relation $\{(x, z, y) \mid x \leq z \leq y\}$. We assume the state sets Q_A and Q_\leq are disjoint.

For $x, y \in L$ and $v \in \Sigma^*$, define $[x, y]_v$ as the set of all $z \in L$ such that $x \leq z \leq y$ and z has prefix v . For $|v| \geq |x|, |y|$ define $I(x, v, y) \subset 2^{Q_A}$ and $J(x) \subset 2^{Q_\leq}$ as follows. $I(x, v, y)$ is the set of all states in Q_A reachable from the initial state ι_A after reading the convolution of (x, v, y) , say $(x \diamond^n, v, y \diamond^m)$ where $n, m \geq 0$ are chosen so that the length of each component is exactly $|v|$. That is define $I(x, v, y) := \Delta_A(\iota_A, (x, v, y)^\diamond)$. Similarly, define $J(v) := \Delta_\leq(\iota_\leq, (v, v)^\diamond)$. Write $K(x, v, y)$ for the ordered pair $(I(x, v, y), J(v))$.

Now if $K(x, v, y) = K(x', v', y')$, then $[x, y]_v$ is isomorphic to $[x', y']_{v'}$ via the map $vw \mapsto v'w$ for $w \in \Sigma^*$. Indeed, the domains are isomorphic since $vw \in [x, y]_v$ if and only if

$$\Delta_A(\Delta_A(\iota_A, (x, v, y)^\diamond), w) \cap F_A \neq \emptyset$$

if and only if

$$\Delta_A(\Delta_A(\iota_A, (x', v', y')^\diamond), w) \cap F_A \neq \emptyset$$

if and only if $v'w \in [x', y']_{v'}$.

The map preserves the ordering since for $w_1, w_2 \in \Sigma^*$ such that $vw_1, vw_2 \in [x, y]_v$ and $v'w_1, v'w_2 \in [x', y']_{v'}$ we have $vw_1 \leq vw_2$ if and only if

$$\Delta_\leq(\Delta_\leq(\iota_\leq, (v, v)^\diamond), (w_1, w_2)^\diamond) \cap F_\leq \neq \emptyset$$

if and only if

$$\Delta_\leq(\Delta_\leq(\iota_\leq, (v', v')^\diamond), (w_1, w_2)^\diamond) \cap F_\leq \neq \emptyset$$

if and only if $v'w_1 \leq v'w_2$.

Hence then number of isomorphism types of the form $[x, y]_v$ for $|v| \geq |x|, |y|$ is bounded by the number of distinct sets $K(x, v, y)$ which is at most $2^{Q_A + Q_\leq}$, denoted by d . In particular there are at most d many VD_* -ranks among such intervals of the form $[x, y]_v$. Now let $[x, y]$ be a closed interval of \mathcal{L} . Set $n = \max\{|x|, |y|\}$ and partition $[x, y]$ into the set $[x, y] \cap \Sigma^{<n}$ and the finitely many sets of the form $[x, y]_v$ where $|v| = n$. By Proposition 3.2, one of these intervals has the same VD_* -rank as the interval $[x, y]$. Suppose the VD -rank of $[x, y]$ is greater than d . By Proposition 3.3 there would be more than d intervals of different VD - and hence VD_* -ranks. So among intervals of the form $[x, y]_v$, where $|v| \geq |x|, |y|$, there would be more than d many VD_* -ranks, a contradiction. We conclude that the VD -rank of $[x, y]$ is at most d . So for every $x, y \in L$, $c^d(x) = c^d(y)$ so $VD(\mathcal{L}) \leq d$ as required. \square

As a corollary of the theorem just proved we derive the following result for all automatic linear orderings:

Theorem 3.5 *If \mathcal{L} is an automatic linear order, then its FC-rank is finite.*

Proof Let $(Q_\leq, \iota_\leq, \Delta_\leq, F_\leq)$ be the 2-tape automaton recognising the ordering of \mathcal{L} . Let $(Q_A, \iota_A, \Delta_A, F_A)$ be the 3-tape automaton recognising the definable relation $\{(x, z, y) \mid x \leq z \leq y\}$. Now consider an interval $[a, b]$ contained in the maximal scattered linear order containing a given element x of \mathcal{L} . Consider the constant d defined in the proof of the previous theorem. Note that this constant does not depend on the interval $[a, b]$ chosen. Therefore the FC-rank of the interval is at most d . We conclude that the FC-rank of the maximal scattered order is at most d , and so the FC-rank of \mathcal{L} is at most d . \square

The results above can now be applied to show that the isomorphism problem for automatic ordinals is decidable. Contrast this with the fact that the isomorphism problem for computable ordinals is Π_1^1 -complete. Recall that by Cantor's normal form theorem if α is an ordinal then it can be uniquely decomposed as $\omega^{\alpha_1}n_1 + \omega^{\alpha_2}n_2 + \dots + \omega^{\alpha_k}n_k$, where $\alpha_1, \alpha_2, \dots, \alpha_k$ are ordinals satisfying $\alpha_1 > \alpha_2 > \dots > \alpha_k$ and k, n_1, n_2, \dots, n_k are natural numbers. Our proof of deciding the isomorphism problem for automatic ordinals based on the fact that the Cantor normal form can be extracted from automatic presentations of ordinals.

Theorem 3.6 If α is an automatic ordinal then its normal form is computable from an automatic presentation of α .

Proof The automatic presentation for α is given by a regular set $R \subseteq \Sigma^*$ for some alphabet Σ and an automaton for the ordering \leq_{ord} on R . Recall that the unknown ordinal represented by (R, \leq_{ord}) is of the form $\alpha = \omega^m n_m + \omega^{m-1} n_{m-1} + \dots + \omega^2 n_2 + \omega n_1 + n_0$ where $m, n_m, n_{m-1}, \dots, n_1, n_0$ are natural numbers. Now one can compute the values m, n_0, n_1, \dots by the following algorithm.

1. **Input** the presentation (R, \leq_{ord}) .

2. Let $D = R, m = 0, n_m = 0$.

3. **While** $D \neq \emptyset$ **Do**

4. **If** D has a maximum u

Then Let $n_m = n_m + 1$, let $D = D - \{u\}$.

Else Let $L \subseteq D$ be the subset of limit ordinals in D ; that is L is the set of all $x \in D$ with no predecessor in D . Replace D by L , let $m = m + 1$, let $n_m = 0$.

5. **End While**

6. **Output** the formula

$$\omega^m n_m + \omega^{m-1} n_{m-1} + \dots + \omega^2 n_2 + \omega n_1 + n_0$$

using the current values of m, n_0, \dots, n_m .

Since the first order theory of an automatic structure is decidable, each step in the algorithm is computable. Removing the maximal element from D reduces the ordinal represented of D by 1 while the corresponding n_m is increased by 1. Replacing D by the set of its limit ordinals is like dividing the ordinal represented by D by ω , so that the next coefficient can start to be computed. Based on this it is easy to verify that the algorithm computes the coefficients n_0, n_1, \dots in this order. The algorithm eventually terminates since m is bounded by the finite bound on the VD -rank of the ordinal. \square

The following corollary is immediate.

Corollary 3.7 The isomorphism problem for automatic ordinals is decidable.

Compare this with the fact that the isomorphism problem for automatic structures, and even permutation structures, is not decidable [3],[8]. We do not know if the isomorphism problem for automatic linear orders is decidable.

We would like to say a few words on the problem of characterising automatic linear orders. We already have a characterisation of the isomorphism types of linear orders presentable over a unary domain.

Theorem 3.8 [1, 12] A linear ordering is automatically presentable over a unary alphabet if and only if it is a finite sum of linear orders of VD -rank ≤ 1 .

The non-unary case is far more involved. A full characterisation of the automatic scattered linear orders would further refine Theorem 3.5. For example $\mathcal{L} = \Sigma_{i \in \mathbb{N}} (\mathbb{Z} + \mathbf{n}_i)$, where $f(i) = n_i$ is function from ω into ω , has FC -rank 2. Note that if f is a non-recursive function then \mathcal{L} is not automatic; otherwise the decidability of \mathcal{L} could be used to compute f . The following examples indicate the complexities involved in the general case.

Example 3.9 [8] Let \mathcal{E} is an automatic equivalence structure, with finite equivalence classes $\{B_i\}$ for $i \in \omega$. Then the linear order $\mathcal{L}_{\mathcal{E}} = \Sigma \{\eta + \mathbf{n}_i \mid i \in \omega\}$ of F -rank 2, where n_i is the cardinality of B_i , is automatically presentable.

Hence a characterisation of automatic equivalence structures would follow from one of automatic linear orders. But automatic equivalence structures already exhibit the following behaviour.

Example 3.10 [8] Consider equivalence structures $\mathcal{E}_1 = (\{0, 1\}^*, E)$ and $\mathcal{E}_2 = (0^* 1^*, E)$, where $E(x, y)$ if $|x| = |y|$ and x, y belong to the same domain. In \mathcal{E}_1 for each n there is a unique equivalence class of size 2^n . In \mathcal{E}_2 for each n there is a unique equivalence class of size $(n+1)(n+2)/2$. In general, for every function f that is either a polynomial whose coefficients are positive integers, or an exponential function k^{an+b} , where $k \geq 2$ and a, b are fixed positive integers, there exists an automatic equivalence relation \mathcal{E} with a unique finite equivalence class of size $f(n)$ for every $n \geq 1$.

4. Automatic trees

A tree $\mathcal{T} = (T, \preceq)$ is a partial order that has a least element r , called the root, and in which $\{y \in T \mid y \preceq x\}$ is a finite linear order for each $x \in T$. Write $x \parallel y$ if $x \not\preceq y$ and $y \not\preceq x$. The set $S(x)$ of immediate successors of x is defined as $S(x) = \{y \in \mathcal{T} \mid x \prec y \wedge \forall z (x \preceq z \preceq y \Rightarrow z \in \{x, y\})\}$. A tree \mathcal{T} is **finitely branching** if $S(x)$ is finite for each $x \in T$. A **path** of a tree (T, \preceq) is a subset $P \subseteq T$ which is linearly ordered, closed downward (that is, whenever $y \in P$ and $x \preceq y$ then $x \in P$), and maximal (under set-theoretic inclusion) with these properties.

Let Σ be the underlying alphabet and let \leq_u be the length-lexicographic order of Σ^* where $x \leq_u y$ if either $|x| < |y|$ or $|x| = |y|$ but x lexicographic before y . For example, $\lambda <_u 0 <_u 1 <_u 00 <_u 01 <_u \dots$ in the case that $\Sigma = \{0, 1\}$. Thus, if tree \mathcal{T} is automatic over the alphabet Σ , and hence $T \subseteq \Sigma^*$, then the length-lexicographic order

on Σ^* is inherited by each set $S(x)$. This permits us to talk about the first, second, third, ... successor of x .

Trees have been studied intensively. If one considers Turing machines instead of finite automata, there are trees which have infinite paths, but no hyperarithmetic one, in particular no recursive infinite path. Furthermore, even finitely branching trees might have infinite paths but none of them is recursive. In contrast to this, the next result states that every automatic tree, not necessarily finitely branching, either has a regular infinite path or does not have an infinite path at all.

Theorem 4.1 *If an automatic tree has an infinite path, then it has a regular infinite path.*

Proof Let $\mathcal{T} = (T, \preceq)$ be an automatic tree with some infinite path where T is a regular subset of Σ^* for a finite alphabet Σ . The proof consists of the following two parts.

1. Given an automatic tree $\mathcal{T} = (T, \preceq)$, the set T' of all nodes in T which are on an infinite path is regular.
2. There is a definable infinite path P on T' in the language expanded by the automatic length-lexicographic order. Hence P is regular.

1. To prove that T' is regular it suffices to show that:

- There is an alphabet Δ , a function $\pi : \Delta \rightarrow \Sigma_\diamond$ and a Büchi automaton \mathcal{B} such that $a_0 \dots a_n \in T'$ if and only if there is an infinite sequence $c_0 c_1 \dots \in \Delta^\infty$ accepted by \mathcal{B} and $a_0 = \pi(c_0), a_1 = \pi(c_1), \dots, a_n = \pi(c_n), \diamond = \pi(c_{n+1})$.

Choose the alphabet Δ as $\Sigma_\diamond \times \Sigma$ where π is the projection from Δ onto its first coordinate, that is, $\pi(a, b) = a$. Say that a word x is on $c_0 c_1 \dots$, where each c_i is $(a_i, b_i) \in \Sigma_\diamond \times \Sigma$, iff there exist $m, n \in \mathbb{N}$ such that

either $m = 0$, $x = a_0 a_1 \dots a_n$ and $a_{n+1} = \diamond$

or $n \geq m > 0$, $x = b_0 b_1 \dots b_{m-1} a_m a_{m+1} \dots a_n$, $a_{m-1} = \diamond$ and $a_{n+1} = \diamond$.

In the first case we say that x is the *first word* on $c_0 c_1 \dots$. Consider the set of all sequences $(a_0, b_0)(a_1, b_1) \dots \in \Delta$ such that there are infinitely many words on the sequence and the words on the sequence generate an infinite path of T . More formally,

- $\exists^\infty n (a_n = \diamond)$;
- if y, z are on $(a_0, b_0)(a_1, b_1) \dots$ and $|y| \leq |z|$ then $y \preceq z$ and $y, z \in T$.

There is a Büchi automaton \mathcal{B} accepting such sequences because the orderings \preceq and length-comparison are automatic and T is regular.

We prove that $x \in T'$ if and only if x is the first word on some sequence $c_0 c_1 \dots$ satisfying the two conditions. The reverse implication is clear. For the forward implication let $x \in T$ be given and P be an infinite path witnessing that $x \in T'$. Define the sequences $a_0 a_1 \dots$ and $b_0 b_1 \dots$ described below.

1. **Choose** n, a_0, a_1, \dots, a_n such that $x = a_0 a_1 \dots a_n$.
Let $a_{n+1} = \diamond$.
2. **Let** $m = 0$. **Let** $y = x$.
3. **Find** $b_m b_{m+1} \dots b_{n+1}$ such that infinitely many nodes in P extend $b_0 b_1 \dots b_{n+1}$ as strings.
4. **Update** $m = n + 2$.
5. **Find** a new value for n and $a_m a_{m+1} \dots a_n$ such that $n \geq m$, the node $z = b_0 b_1 \dots b_{m-1} a_m a_{m+1} \dots a_n$ is in P and $y \preceq z$. **Let** $a_{n+1} = \diamond$.
6. **Let** $y = z$. **Go to** 3.

Note that it is an invariant of the construction that whenever the algorithm comes to Step 3, either $m = 0$ or infinitely many nodes in P extend the string $b_0 b_1 \dots b_{m-1}$. As there are only finitely many choices for the new part $b_m b_{m+1} \dots b_{n+1}$, one can choose this part such that still infinitely many nodes in P extend $b_0 b_1 \dots b_{n+1}$ as a string. In Step 4, m is chosen such that the precondition of Step 3 holds again and b_m is the first of the b -symbols not yet defined. For every $y \in P$ it holds that almost all nodes z in P satisfy $y \preceq z$. Furthermore, for every finite length l , almost all nodes in P are represented by strings longer than l . Thus one can find a node z as specified in Step 5 and the algorithm runs forever defining the infinite sequence $(a_0, b_0)(a_1, b_1) \dots$ in the limit. In particular, such a sequence exists. It is not required that the sequence can be constructed effectively since the path P might not even be recursive.

2. Now we give a first order definition of the lexicographically least infinite path. Recall that the length-lexicographic order \leq_{ll} on Σ^* is automatic and therefore one can add \leq_{ll} to the structure \mathcal{T}' without losing the property that it is an automatic structure. Now define the leftmost infinite path P with respect to the length-lexicographic order of the successors of any node. P contains those nodes x for which every $y \prec x$ satisfies that $\forall z, z' \in S(y) [z \preceq x \Rightarrow z \leq_{ll} z']$. This means, that the unique node $z \in S(y)$ which is below x is just the length-lexicographically least element of $S(y)$. Since the length-lexicographic ordering of Σ^* is a well-ordering (of type ω), this minimum always exists. \square

Remark 4.2 In the case that the automatic tree $\mathcal{T} = (T, \preceq)$ is infinite and finitely branching, one can simplify the first part of the proof. The reason is that T' is definable by formula $x \in T' \Leftrightarrow (\exists^\infty y) [x \preceq y]$. If there are infinitely many nodes above x , x is by König's Lemma on an infinite path. If there are only finitely many nodes above x , these nodes trivially cannot contain an infinite path. The second part is proven in the same way as above.

From Theorem 4.1, we see that if an automatic tree has *finitely* many infinite paths, then each is regular. We generalise this to trees with *countably* many infinite paths (Theorem 4.4). The proof relies on associating the Kleene-Brouwer-ordering with trees.

Definition 4.3 [13] Let (T, \preceq) be a tree and \leq_{ll} be the lexicographic order induced by the presentation of T as a subset of Σ^* . Let x, y be nodes on T . Then $x \leq_{kb} y$ iff either $y \preceq x$ or there are u, v, w such that $v, w \in S(u)$, $v \preceq x$, $w \preceq y$ and $v \leq_{ll} w$.

Theorem 4.4 If an automatic tree has countably many infinite paths then every infinite path in it is regular.

Proof Let (T_1, \preceq) be an automatic tree with countably many infinite paths. Let Σ be the alphabet and \leq_{ll} be the length-lexicographic ordering. Furthermore \leq_{kb} is the Kleene-Brouwer ordering derived from \leq_{ll} and T_1 as defined in Definition 4.3. Inductively, for $n = 1, 2, \dots$, let

$$\begin{aligned} T'_n &= \{x \in T_n \mid \exists \text{ inf. path } P \text{ of } T_1 \\ &\quad (x \in P \wedge P \cap T_n \text{ is infinite})\}; \\ T'_{n+1} &= \{x \in T'_n \mid \forall y \in T'_n \exists z \in T'_n \\ &\quad (y \leq_{kb} x \vee x <_{kb} z <_{kb} y)\}. \end{aligned}$$

The set T'_n contains those nodes which are on an infinite path of T_1 which has an infinite intersection with T_n . The set T'_{n+1} contains those nodes of T'_n which are the infimum of the properly above nodes in T'_n . The structures $(T_n, \preceq, \leq_{kb})$ and $(T'_n, \preceq, \leq_{kb})$ are automatic iff the corresponding sets T_n and T'_n are regular subsets of Σ^* . The set T_1 is regular. If T_n is regular, so is T'_n by the construction in the first part of the proof of Theorem 4.1. T'_{n+1} is obtained from T'_n using a first-order formula and therefore also regular. So one has by induction that $(T_1, T'_1, T_2, T'_2, \dots, \preceq, \leq_{kb})$ is an automatic structure.

Now it is first proven that there is an n for which T_n is finite. Let $c_F^1(A, x) = c_F^1(A, y)$ denote that there are only finitely many elements of A between x and y with respect to \leq_{kb} , for $m \geq 1$ the notion $c_F^m(A, x) = c_F^m(A, y)$ is the iterated versions of it. If $x, y \in T'_n$ and $c_F(T'_n, x) = c_F(T'_n, y)$, then there are only finitely many elements between them and one of the numbers x, y cannot be the limit inferior of an infinite descending chain in (T'_n, \leq_{kb}) . Thus at most one of the

elements x, y is in T_{n+1} . Iterating the argument, it follows that for every $x, y \in T_1$ with $c_F^n(T_1, x) = c_F^n(T_1, y)$, at most one of the elements x, y is in T_{n+1} . Since the structure (T_1, \leq_{kb}) is an automatic linear ordering, its FC-rank is finite and thus some T_n contains at most one element. In particular $T_n \cap P$ is finite for all infinite paths P of T_1 .

So there is, for every infinite path P of T_1 , a maximal number n such that $T_n \cap P$ is infinite. It follows from the definition of T'_n that $T'_n \cap P = T_n \cap P$ and so, $T'_n \cap P$ is also infinite. Now take a node $x \in T'_n \cap P$ such that no node $z \succeq x$ is in $T_{n+1} \cap P$. Let y be the least element of $T'_n \cap P$ with respect to \preceq satisfying $x \prec y$.

Assume that there would be an another $z \succ x$ in T'_n which satisfies $y <_{kb} z <_{kb} x$. Then $y \parallel z$. But since $z \in T'_n$ there is an infinite path P' of T_1 such that $P' \cap T'_n$ is infinite and contains z . So there is a node $z' \succ z$ in $P' \cap T'_n$. This node satisfies $y <_{kb} z' <_{kb} z$. Therefore y is in T'_n the infimum of the nodes $\{z \in T'_n \mid y <_{kb} z\}$ and y would be in T_{n+1} in contradiction to the choice of n, x, y .

Thus it holds for all $y \in P \cap T'_n$ and $z \in T'_n$, whenever $y \leq_{kb} z \leq_{kb} x$ then $z \in P$. Furthermore, every $y \in P \cap T'_n$ with $x \preceq y$ satisfies that the set $\{z \mid y \leq_{kb} z \leq_{kb} x\}$ is finite. So the first-order definable set $P_{x,n} = \{y \succeq x \mid y \in T'_n \wedge \neg \exists^\infty z (y \leq_{kb} z \leq_{kb} x)\}$ is equal to $\{y \succeq x \mid y \in P \cap T'_n\}$. It follows that P is the downward closure with respect to \preceq in T_1 of $P_{x,n}$ and P is regular. \square

In Section 5 it is shown that every automatic tree has finite Cantor-Bendixson rank. This would permit to simplify the constructions of the sets T'_1, T_2, T'_2, \dots which then are subtrees still satisfying that almost all of these sets are empty.

$$\begin{aligned} T'_n &= \{x \in T_n \mid \exists \text{ inf. path } P \text{ of } T_1 (x \in T_n)\}; \\ T'_{n+1} &= \{x \in T'_n \mid \exists y, z \in T'_n (x \preceq y \wedge x \preceq z \wedge y \parallel z)\}. \end{aligned}$$

If T_1 is furthermore finitely branching, the first condition is equivalent to

$$T'_n = \{x \in T_n \mid \exists^\infty y \in T_n (x \preceq y)\}$$

so that all trees T_n, T'_n are first-order definable in the language (T_1, \preceq) . For every infinite path P of T_1 there is a maximal n such that P is on T_n . If a is a node on P which is sufficiently large than all $b \in T_n$ satisfy that $b \in P$ iff $a \preceq b \vee b \preceq a$. Only finitely many of the trees, say T_1, T_2, T_3, T_4 , have infinite paths and the formula

$$\begin{aligned} \phi(a, b) &= (a \in T'_1 \wedge a \notin T_2 \wedge b \in T'_1 \wedge (b \preceq a \vee a \preceq b)) \\ &\vee (a \in T'_2 \wedge a \notin T_3 \wedge b \in T'_2 \wedge (b \preceq a \vee a \preceq b)) \\ &\vee (a \in T'_3 \wedge a \notin T_4 \wedge b \in T'_3 \wedge (b \preceq a \vee a \preceq b)) \\ &\vee (a \in T'_4 \wedge a \notin T_5 \wedge b \in T'_4 \wedge (b \preceq a \vee a \preceq b)) \end{aligned}$$

tells for every infinite path P and almost every $a \in P$ which nodes b are on P . So $\{b \mid \phi(a, b)\}$ is either an infinite path of T_1 or empty.

In the case that (T_1, \preceq) is infinite branching, one modifies the tree. Recall that the set $S(x)$ of immediate successors of $x \in T$:

$$y \in S(x) \Leftrightarrow x \prec y \wedge \forall z(z \not\prec y \vee x \not\prec z).$$

Furthermore, let \leq_{ll} be the length-lexicographic ordering on the underlying set Σ^* . Now let r be the root of the tree (T_1, \preceq) and define the new ordering \preceq' by letting $x \preceq' y$ if and only if

$$x = r \vee \exists v, w \in T_1 (x, w \in S(v) \wedge x \leq_{ll} w \wedge w \preceq y)$$

The tree (T_1, \preceq') is finitely branching. Every infinite branch P' of (T_1, \preceq') are generated either by an infinite branch P of (T_1, \preceq) or by an infinite set of the form $S(x)$ where the set $S(x)$ refers to the immediate successors of an $x \in T_1$ with respect to \preceq . Let ϕ' be the formula for the infinite branches of (T_1, \preceq') . If P' is an infinite branch of T_1 which is generated by P , then an $x \in P'$ is also in P iff there is an $y \in P'$ such that $x \prec y$. Furthermore if P' is generated by a set $S(u)$ then almost all $x \in P'$ do not have an $y \in P'$ with $x \prec y$ since these x are in $S(u)$. Thus the following formula ϕ derived from ϕ' has the desired properties:

$$\phi(a, b) = \phi'(a, b) \wedge \exists^\infty x \exists y (\phi(a, x) \wedge \phi(a, y) \wedge b \preceq x \preceq y).$$

Note furthermore, that the quantification “ \exists^∞ ” enforces as a side-effect that $\phi(a, b)$ defines only infinite sets and that thus one has the following result.

Theorem 4.5 *If (T, \preceq) is an automatic tree with countably many infinite paths, then there is a formula ϕ such that the sets $P_a = \{b : \phi(a, b)\}$ satisfies the following conditions:*

- If P_a is not empty, then P_a is an infinite path of (T, \preceq) containing a ;
- Every infinite path of (T, \preceq) is equal to some set P_a .

The option that P_a can be empty in the definition above cannot be avoided since a might be in T but not on an infinite path of T .

5. Cantor-Bendixson Rank of Automatic Trees

In this section it is shown that every automatic tree with countably many infinite paths has finite Cantor-Bendixson Rank. For this, we need some additional notation.

We write $[\mathcal{T}]$ for the set of infinite paths of \mathcal{T} and if $x \in T$ and $p = (p_i) \in [\mathcal{T}]$ write $x \prec p$ if $x = p_i$ for some i . Define the extendible part $E(\mathcal{T})$ of \mathcal{T} as $\{x \in T \mid \exists p \in [\mathcal{T}], x \prec p\}$. Note that $E(\mathcal{T}) = \{x \in T \mid \exists^\infty y, x \prec y\}$ if \mathcal{T} is finitely branching. Let $F(\mathcal{T})$ be the domain $\{x \in$

$E(\mathcal{T}) \mid \exists y, z \in E(\mathcal{T}), y, z \succ x \text{ and } y \parallel z\}$. Write $d(\mathcal{T})$ for the tree resulting by restricting \mathcal{T} to $F(\mathcal{T})$. In words, a node $x \in \mathcal{T}$ is in $d(\mathcal{T})$ if and only if there are at least two distinct infinite paths in \mathcal{T} above x . For each ordinal α define the iterated operation $d^\alpha(\mathcal{T})$ inductively as follows.

1. $d^0(\mathcal{T}) = \mathcal{T}$.
2. $d^{\alpha+1}(\mathcal{T})$ is $d(d^\alpha(\mathcal{T}))$.
3. If α is a limit ordinal, then $d^\alpha(\mathcal{T})$ is $\cap_{\beta < \alpha} d^\beta(\mathcal{T})$.

Definition 5.1 [9] *The Cantor-Bendixson Rank of a tree \mathcal{T} (written $CB(\mathcal{T})$) is the least ordinal α such that $d^\alpha(\mathcal{T}) = d^{\alpha+1}(\mathcal{T})$.*

Lemma 5.2 1. $CB(\mathcal{T})$ is a countable ordinal.

2. Suppose \mathcal{T} has countably many infinite paths.

- (a) $\alpha = CB(\mathcal{T})$ is 0 or a successor ordinal and $d^\alpha(\mathcal{T})$ is the empty tree.
- (b) $CB(\mathcal{T}) = 0$ if and only if \mathcal{T} is the empty tree.
- (c) $CB(\mathcal{T}) = 1$ if and only if \mathcal{T} is non-empty and contains at most one infinite path.

Proof For each β let $x_\beta \in d^\beta(\mathcal{T}) \setminus d^{\beta+1}(\mathcal{T})$. Since T is countable, and $\alpha \neq \beta$ implies that $x_\alpha \neq x_\beta$, the set of ordinals β such that $d^\beta(\mathcal{T}) \setminus d^{\beta+1}(\mathcal{T}) \neq \emptyset$ is also countable. Hence its least upper bound, a countable ordinal, say α , is $CB(\mathcal{T})$. If $d^\alpha(\mathcal{T})$ is not the empty tree, then every element of $d^\alpha(\mathcal{T})$ has at least two distinct infinite paths above it. In particular, $d^\alpha(\mathcal{T})$ embeds the full binary tree, so \mathcal{T} has uncountably many infinite paths. Finally suppose $\alpha > 0$ is a limit ordinal. Then $d^\alpha(\mathcal{T})$ is non-empty since the root of \mathcal{T} is in $d^\beta(\mathcal{T})$ for every $\beta < \alpha$. Hence the CB -rank of \mathcal{T} is 0 or a successor ordinal. The last two items follow directly from the definition of d . \square

The next definition associates a linear ordering \mathcal{L}_T with a tree \mathcal{T} .

Definition 5.3 *Let (T, \preceq) a tree and recall that \leq_{ll} is a linear order of type ω on T . Let $x, y \in T$. Then $x <_{lt} y$ if either $x \prec y$ or there are u, v, w such that $v, w \in S(u)$, $v \preceq x$, $w \preceq y$ and $v <_{ll} w$. Write \mathcal{L}_T for the structure $(T, <_{lt})$.*

Then \mathcal{L}_T is a linear ordering and is definable in (\mathcal{T}, \leq_{ll}) .

Theorem 5.4 *The CB -rank of an automatic finitely branching tree with countably many infinite paths is finite.*

Proof Suppose \mathcal{T} is finitely branching with countably many

infinite paths. We now prove that \mathcal{L}_T is scattered and is a finite sum of orders of VD -rank $\leq CB(\mathcal{T})$. In this case, if \mathcal{T} is automatic then so is \mathcal{L}_T , which by theorem 3.4 has finite VD -rank. Then $CB(\mathcal{T})$ must also be finite as required.

We deal with the base cases first. If $CB(\mathcal{T}) = 0$ then \mathcal{T} is the empty tree and so \mathcal{L}_T is the empty linear order, which satisfies the conclusion. If $CB(\mathcal{T}) = 1$ then \mathcal{T} is non-empty and contains at most one infinite path. By the definition of $<_{lt}$, L_T has order type $\mathbf{n}, \omega, \omega + \mathbf{n}$ or $\omega + \omega^*$. In every case it is scattered and a finite sum of orders of VD -rank ≤ 1 .

Now let $\alpha = CB(\mathcal{T}) > 1$. By the previous lemma, $\alpha = \beta + 1$ for some $\beta > 0$. For $x \in T$ let $T(x)$ be the subtree of \mathcal{T} rooted at x , that is $\{y \in T \mid x \preceq y\}$. Define $X = \{x \in T \mid CB(T(x)) = \alpha\}$. Then $X \neq \emptyset$ since the root of \mathcal{T} is in X . Further $X = d^\beta(\mathcal{T})$. Indeed if $x \in X$ then in particular $x \in d^\beta(T(x))$ and so $x \in d^\beta(\mathcal{T})$. Conversely if $x \notin X$ then $d^\gamma(T(x)) = \emptyset$ for some $\gamma \leq \beta$. Hence $x \notin d^\gamma(\mathcal{T})$ and in particular $x \notin d^\beta(\mathcal{T})$.

Hence $CB(X) = 1$ and so X contains at most one infinite path. For $x \in X$, consider an immediate successor of x in \mathcal{T} , that is not itself in X , say $y \in S(x) \setminus X$. Then by definition of X , $CB(T(y)) \leq \beta$. By the induction hypothesis $L_{T(y)}$ is scattered and is a finite sum of orders of VD rank $\leq \beta$. If X is finite then L_T is a finite sum of the $L_{T(y)}$'s, of which there are finitely many since T is finitely branching. Hence L_T is scattered and satisfies the conclusion. If X is infinite, let (x_i) be the unique infinite path in X . Then from the definition of $<_{lt}$,

$$L_T = \sum_{i \in \omega} (x_i + L_i) + \sum_{j \in \omega^*} R_j,$$

where every L_i or R_j is some $T(y)$. Hence L_T is scattered and is a finite sum of orders of VD -rank $\leq \beta + 1$ as required. \square

Theorem 5.5 *The CB -rank of an automatic tree with countably many infinite paths is finite.*

Proof As in the text before Theorem 4.5, one defines from the given tree (T, \preceq) the tree (T, \preceq') such that $x \preceq' y$ iff

$$x = r \vee \exists v, w \in T (x, w \in S(v) \wedge x \leq_u w \wedge w \preceq y);$$

where r is the root of T , \leq_u the length-lexicographic order and $S(v)$ the set of immediate successors of v with respect to \preceq . The finitely branching tree (T, \preceq') also has countably many infinite paths and therefore finite CB -rank. Let U and U' be the topological spaces of the sets of infinite paths of (T, \preceq) and (T, \preceq') , respectively. Every infinite path P of (T, \preceq) generates an infinite path P' of (T, \preceq') and one can show that the topological spaces U and $\{P' \in U' \mid \exists P \in U (P \text{ generates } P')\}$ are homeomorphic. Thus, the CB -rank of U' (as a topological space) is an upper bound of that of

U and the same holds for the corresponding trees (as the CB -rank of a tree is the same as the CB -rank of the corresponding topological space). Thus, the Cantor-Bendixson rank of the tree (T, \preceq) is also finite. \square

Acknowledgement The authors thank Wolfgang Merkle for useful discussions and the anonymous referees for helpful comments regarding presentation.

References

- [1] A. Blumensath. *Automatic Structures*. Diploma thesis, RWTH Aachen, 1999.
- [2] A. Blumensath and E. Grädel. Automatic structures. In *15th Symposium on Logic in Computer Science (LICS' 00)*, pages 51–62, June 2000.
- [3] A. Blumensath and E. Grädel. Finite presentations of infinite structures : Automata and interpretations. In *Proc. 2nd Int. Workshop on Complexity in Automated Deduction(CIAD)*, 2002.
- [4] C. Delhomme. Non-automaticity of ω^ω , 2001. manuscript.
- [5] C. Delhomme, V. Goranko, and T. Knapik. Automatic linear orderings, 2003. manuscript.
- [6] S. Eilenberg, C. Elgot, and J. Shepherdson. Sets recognised by n -tape automata. *Journal of Algebra*, 13(4):447–464, 1969.
- [7] D. B. H. Epstein, J. W. Cannon, D. F. Holt, S. V. F. Levy, M. S. Paterson, and W. P. Thurston. *Word processing in groups*. Jones and Bartlett, 1992.
- [8] H. Imai, B. Khoussainov, and S. Rubin. Some results on automatic structures. In *17th Symposium on Logic in Computer Science (LICS' 02)*, pages 235–242, July 2002.
- [9] A. Kechris. *Classical Descriptive Set Theory*. Springer-Verlag, 1995.
- [10] B. Khoussainov and A. Nerode. Automatic presentations of structures. *Lecture Notes in Computer Science*, 960:367–392, 1995.
- [11] B. Khoussainov and S. Rubin. Automatic structures: Overview and future directions. *Journal of Automata, Languages and Combinatorics. Selected papers of Weighted Automata : Theory and Applications* (2002).
- [12] B. Khoussainov and S. Rubin. Graphs with automatic presentations over a unary alphabet. *Journal of Automata, Languages and Combinatorics*, 6(4), 2001.
- [13] H. Rogers. *Theory of Recursive Functions and Effective Computability*. McGraw-Hill, 1967.
- [14] J. Rosenstein. *Linear Orderings*. Academic Press, 1982.

Verifying ω -Regular Properties of Markov Chains

Doron Bustan¹, Sasha Rubin², and Moshe Y. Vardi¹

¹ Rice University***

² The University of Auckland

Abstract. In this work we focus on model checking of probabilistic models. Probabilistic models are widely used to describe randomized protocols. A Markov chain induces a probability measure on sets of computations. The notion of correctness now becomes probabilistic. We solve here the general problem of linear-time probabilistic model checking with respect to ω -regular specifications. As specification formalism, we use alternating Büchi infinite-word automata, which have emerged recently as a generic specification formalism for developing model checking algorithms. Thus, the problem we solve is: given a Markov chain \mathcal{M} and automaton \mathcal{A} , check whether the probability induced by \mathcal{M} of $L(\mathcal{A})$ is one (or compute the probability precisely). We show that these problem can be solved within the same complexity bounds as model checking of Markov chains with respect to LTL formulas. Thus, the additional expressive power comes at no penalty.

1 Introduction

In model checking, we model a system as a transition system \mathcal{M} and a specification as a temporal formula ψ . Then, using formal methods, we check whether \mathcal{M} satisfies ψ [7]. One of the most significant developments in this area is the discovery of algorithmic methods for verifying temporal logic properties of *finite-state* systems [25, 19, 6, 35]. This derives its significance both from the fact that many synchronization and communication protocols can be modelled as finite-state programs, as well as from the great ease of use of fully algorithmic methods. Looking at model-checking algorithms more closely, we can classify these algorithms according to two criteria. The first criterion is the type of model that we use – nondeterministic or probabilistic. The second criterion is the specification language.

For nondeterministic models and linear temporal logic (LTL), a close and fruitful connection with the theory of automata over infinite words has been developed [34–36]. The basic idea is to associate with each LTL formula a nondeterministic Büchi automaton over infinite words (NBW) that accepts exactly all the computations that satisfy the formula. This enables the reduction of various decision problems, such as satisfiability and model checking, to known automata-theoretic problems, yielding clean and asymptotically optimal algorithms. Furthermore, these reductions are very helpful for implementing temporal-logic based verification algorithms, cf. [14]. This connection to

*** Supported in part by NSF grants CCR-9988322, CCR-0124077, CCR-0311326, IIS-9908435, IIS-9978135, and EIA-0086264, by BSF grant 9800096, and by a grant from the Intel Corporation.

automata theory can also be extended to languages beyond LTL, such as ETL [36] and μ TL [32].

In this paper we focus on model checking of probabilistic models. Probabilistic models are widely used to describe randomized protocols, which are often used in distributed protocols [5], communication protocols [8], robotics [29], and more. We use Markov chains as our probabilistic model, cf. [31]. A Markov chain induces a probability measure on sets of computations. The notion of correctness now becomes probabilistic: we say here that a program is correct if the probability that a computation satisfies the specification is one (we also discuss a quantitative notion of correctness, where we compute the probability that a computation satisfies the specification). Early approaches for probabilistic model checking of LTL formulas [20, 31] required determinization of NBW, which involves an additional exponential blow-up over the construction of automata from formulas [26], requiring exponential space complexity, unlike the polynomial space complexity of standard model-checking algorithms for LTL, cf. [30].

The exponential gap for probabilistic model checking was bridged in [9, 10], who provided a polynomial-space algorithm, matching the lower bound of [28]. The algorithm in [9, 10] is specialized to LTL (and ETL) specifications, and proceeds by induction on the structure of the formula. An automata-theoretic account of this algorithm was given in [11]. It is shown there that LTL formulas can be translated to a special type of NBW, which they call *separated automata*. (An NBW is separated if every two states that are located in the same strongly connected component have disjoint languages). As with the standard translation of LTL formulas to NBW, the translation to separated automata is exponential. It is then shown in [11] how to model check Markov chains, in nondeterministic logarithmic space, with respect to separated NBW as complemented specification. This yields a polynomial space upper bound for probabilistic model checking of LTL formulas.

The automata-theoretic framework in [11] is very specifically tailored to LTL. As mentioned earlier contrast, the automata-theoretic framework for model checking non-deterministic models is quite general and can also handle more expressive specification languages such as ETL and μ TL. This is not a mere theoretical issue. There has been a major recent emphasis on the development of industrial specification languages. These efforts resulted in several languages [18, 23, 4, 2], culminating in an industrial standard, PSL 1.01 (www.accelera.com). Most of the new languages have the full power of NBW, i.e., they can express all ω -regular languages. Thus, they are strictly more expressive than LTL [37], and, thus, not covered by the framework of [9–11].

In this paper we solve the general problem of probabilistic model checking with respect to ω -regular specifications. As specification formalism, we use alternating Büchi infinite-word automata (ABW); see discussion below. Thus, the problem we solve is: Given a Markov chain \mathcal{M} and an ABW \mathcal{A} , check whether the probability induced by \mathcal{M} of $L(\mathcal{A})$ is one (i.e., whether $P_{\mathcal{M}}(L(\mathcal{A})) = 1$). (A more refined problem is to calculate the probability precisely, see Appendix D.)

The motivation for using ABWs as a specification formalism is derived from recent developments in the area of linear specification languages. First, ABWs have been used as an intermediate formalism between LTL formulas and nondeterministic Büchi word automata (NBW). As shown in [12, 13], one can exploit the linear translation from LTL

formulas to ABWs ([33]) for an early minimization, before the exponential translation to NBW. Second, not only can logics such as ETL and μ TL be easily translated to ABW, but also most of the new industrial languages can be translated to ABW. Furthermore, for some of them efficient such translations are known (cf. [1]). Thus, ABW can serve as a generic specification formalism for developing model checking algorithms. Note that applying the techniques of [9, 10] to ABW specifications requires converting them first to NBW at an exponential cost, which we succeed here in avoiding.

We present here an algorithm for model checking of Markov chains, using ABWs as specifications. The space complexity of the algorithm is polylogarithmic in \mathcal{M} and polynomial in \mathcal{A} . The linear translation of LTL to ABW implies that this complexity matches the lower bound for this problem.

As in [10, 11], our algorithm uses the subset construction to capture the language of every subset of states of \mathcal{A} (an infinite word w is in the language of a set Q of states if $w \in L(s)$ for every state $s \in Q$ and $w \notin L(s)$ for every $s \notin Q$). While for LTL, a straightforward subset construction suffices, this is not the case for ABW. A key technical innovation of this paper is our use of *two* nondeterministic structures that correspond to the alternating automaton \mathcal{A} to capture the language of every set of automaton states. The first nondeterministic structure is an NBW \mathcal{A}_f called the *full automaton*, and the second a “slim” version of the full automaton without accepting conditions, which we call the *local transition system* T_A . Every state q of \mathcal{A}_f and T_A corresponds to a set Q of states in \mathcal{A} . While it is possible, however, that a several states of \mathcal{A}_f correspond to the same set of states of \mathcal{A} , every state of T_A corresponds to a unique set of states of \mathcal{A} . The model-checking algorithm make use of the products G and G_f of the Markov chain M with T_A and \mathcal{A}_f , respectively.

2 Preliminaries

2.1 Automata

Definition 1. A nondeterministic Büchi word automaton (NBW) is $\mathcal{A} = \langle \Sigma, S, S_0, \delta, F \rangle$, where Σ is a finite alphabet, S is a finite set of states, $\delta : S \times \Sigma \rightarrow 2^S$ is a transition function, $S_0 \subseteq S$ is a set of initial states, and $F \subseteq S$ is a set of accepting states.

Let $w = w_0, w_1, \dots$ be an infinite word over Σ . For $i \in \mathbb{N}$, let $w^i = w_i, w_{i+1}, \dots$ denote the suffix of w from its i 'th letter. A sequence $\rho = s_0, s_1, \dots$ in S^ω is a *run* of \mathcal{A} over an infinite word $w \in \Sigma^\omega$, if $s_0 \in S_0$ and for every $i > 0$, we have $s_{i+1} \in \delta(s_i, w_i)$. We use $\text{inf}(\rho)$ to denote the set of states that appear infinitely often in ρ . A run ρ of \mathcal{A} is *accepting* if $\text{inf}(\rho) \cap F \neq \emptyset$. An NBW \mathcal{A} accepts a word w if \mathcal{A} has an accepting run over w . We use $L(\mathcal{A})$ to denote the set of words that are accepted by \mathcal{A} . For $s \in S$, we denote by $\mathcal{A}^{(s)}$ the automaton \mathcal{A} with a single initial state s . We write $L(s)$ (the language of s) for $L(\mathcal{A}^{(s)})$ when \mathcal{A} is clear from the context.

Before we define an alternating Büchi word automaton, we need the following definition. For a given set X , let $\mathcal{B}^+(X)$ be the set of positive Boolean formulas over X (i.e., Boolean formulas built from elements in X using \wedge and \vee), where we also allow the formulas **true** and **false**. Let $Y \subseteq X$. We say that Y *satisfies* a formula $\theta \in \mathcal{B}^+(X)$ if the truth assignment that assigns *true* to the members of Y and assigns *false* to the members of $X \setminus Y$ satisfies θ .

Example 1. The sets $\{s_1, s_3\}$ and $\{s_1, s_4\}$ both satisfy the formula $(s_1 \vee s_2) \wedge (s_3 \vee s_4)$, while the set $\{s_1, s_2\}$ does not satisfy this formula. \square

Definition 2. A tree is a set $X \subseteq \mathbb{N}^*$, such that for $x \in \mathbb{N}^*$ and $n \in \mathbb{N}$, if $xn \in X$ then $x \in X$.

We call the node ϵ the root of the tree. A node x is the *parent* of another node t and t is a *child* of x if t is of the form xn . The *level* of a node x , denoted $|x|$, is its distance from the root ϵ ; in particular, $|\epsilon| = 0$. A *branch* $\beta = x_0, x_1, \dots$ of a tree is a maximal sequence of nodes such that x_0 is the root ϵ and x_i is the parent of x_{i+1} for all $0 < i < |\beta|$. Note that β can be finite or infinite. A Σ -labelled tree, for a finite alphabet Σ , is a pair (τ, \mathcal{T}) , where τ is a tree and \mathcal{T} is a mapping from the nodes of τ to Σ that assigns to every node of τ a label in Σ . We often consider \mathcal{T} as the labelled tree. A branch $\beta = x_0, x_1, \dots$ of τ defines a word $\mathcal{T}(\beta) = \mathcal{T}(x_0), \mathcal{T}(x_1), \dots$ consisting of the sequence of labels along the branch.

Definition 3. An alternating Büchi word automaton (ABW) is $\mathcal{A} = \langle \Sigma, S, s_0, \delta, F \rangle$, where Σ, S , and F are as in NBW, $s_0 \in S$ is a single initial state, and $\delta : S \times \Sigma \rightarrow \mathcal{B}^+(S)$ is a transition function.

Because of the universal choice in alternating transitions, a run of an alternating automaton is a tree rather than a sequence. A run of \mathcal{A} on an infinite word $w = w_0, w_1, \dots$ is a (possibly infinite) S -labelled tree τ such that $\mathcal{T}(\epsilon) = s_0$ and the following holds:

if $|x| = i$, $\mathcal{T}(x) = s$, and $\delta(s, w_i) = \theta$, then x has k children $x \cdot 1, \dots, x \cdot k$,
for some $k \leq |S|$, and $\{\mathcal{T}(x \cdot 1), \dots, \mathcal{T}(x \cdot k)\}$ satisfies θ .

The run τ is *accepting* if every infinite branch in τ includes infinitely many labels in F . Note that the run can also have finite branches; if $|x| = i$, $\mathcal{T}(x) = s$, and $\delta(s, w_i) = \text{true}$, then x need not have children.

We define the projected graph of ABW $\mathcal{A} = \langle \Sigma, S, \delta, s_0, F \rangle$ as a graph $\langle S, E \rangle$ where (s_1, s_2) is an edge in E if s_2 appears in $\delta(s_1, \sigma)$ for some $\sigma \in \Sigma$.

Definition 4. An alternating weak word automaton (AWW) is an ABW $\mathcal{A} = \langle \Sigma, S, s_0, \delta, F \rangle$ such that for every strongly connected component C of the projected graph of \mathcal{A} , either $C \subseteq F$ or $C \cap F = \emptyset$.

Lemma 1. [17] Let \mathcal{A} be an ABW. Then there exists an AWW \mathcal{A}_w such that $L(\mathcal{A}) = L(\mathcal{A}_w)$ and the size of \mathcal{A}_w is quadratic in the size of \mathcal{A} . Furthermore, \mathcal{A}_w can be constructed in time quadratic in the size of \mathcal{A} .

Given two AWW \mathcal{A}_1 and \mathcal{A}_2 , we can construct AWW for $\Sigma^\omega \setminus L(\mathcal{A}_1)$, $L(\mathcal{A}_1) \cap L(\mathcal{A}_2)$, and $L(\mathcal{A}_1) \cup L(\mathcal{A}_2)$, which are linear in their size, relative to \mathcal{A}_1 and \mathcal{A}_2 [24].

Next, we present a translation of ABWs into NBWs. Both the translation and its correctness are derived directly from [22]. We say that a tuple $(Q_1, Q_2) \subseteq 2^{S \times S}$ is consistent with respect to \mathcal{A} if $Q_2 \subseteq Q_1 \setminus F$.

Definition 5. Given an ABW $\mathcal{A} = \langle \Sigma, S, s_0, \delta, F \rangle$, we define $N(\mathcal{A})$ as the NBW $\mathcal{A}_n = \langle \Sigma, S_n, (\{s_0\}, \{s_0\} \setminus F), \delta_n, F_n \rangle$, such that:

- S_n is the set of consistent tuples with respect to \mathcal{A} .
- (Q'_1, Q'_2) is in $\delta_n((Q_1, Q_2), \sigma)$ iff $Q'_1 \models \bigwedge_{s \in Q_1} \delta(s, \sigma)$ and either:
 - $Q_2 = \emptyset$ and $Q'_2 = Q'_1 \setminus F$, or
 - $Q_2 \neq \emptyset$ and there exists a set $Y_2 \subseteq Q'_1$ such that $Y_2 \models \bigwedge_{s \in Q_2} \delta(s, \sigma)$, and $Q'_2 = Y_2 \setminus F$.
- $F_n = \{(Q_1, Q_2) \in S_n \mid Q_2 = \emptyset\}$.

Theorem 1. [22] Let Q_1 be a subset of S , and let Q_2 be a subset of $Q_1 \setminus F$. Then $\cap_{s \in Q_1} L(\mathcal{A}^{(s)}) = L(\mathcal{A}_n^{(Q_1, Q_2)})$.

2.2 Markov chains

We model probabilistic systems by finite *Markov chains*. The basic intuition is that transitions between states are governed by some probability distribution.

Definition 6. A Markov chain is a tuple $\mathcal{M} = \langle X, P_T, P_I \rangle$ such that X is a set of states, $P_T : (X \times X) \rightarrow [0, 1]$ is a transition probability distribution that assigns to every transition (x_1, x_2) its probability. P_T satisfies that for every $x_1 \in X$ we have $\sum_{x_2 \in X} P_T(x_1, x_2) = 1$. $P_I : X \rightarrow [0, 1]$ is an initial probability distribution that satisfies $\sum_{x \in X} P_I(x) = 1$.

We denote by $\mathcal{M}^{(x)}$ the Markov chain \mathcal{M} with P_I that maps x to 1. Sometimes we consider a Markov chain as a graph $\langle X, E \rangle$ where $(x_1, x_2) \in E$ iff $P_T(x_1, x_2) > 0$. For an alphabet $\Sigma = 2^{\text{AP}}$, let $V : X \rightarrow \Sigma$ be a labelling function, then each path ρ in X^* or in X^ω in \mathcal{M} is mapped by V to a word w in Σ^* or in Σ^ω respectively. For simplicity we assume that $\Sigma = X$ and that $V(x) = x$ for every $x \in X$, this simplification does not change the complexity of verifying the Markov chain [10]. Note, that every infinite path of \mathcal{M} is a word in X^ω but the converse does not necessarily hold.

As in the standard theory of Markov processes (see [15, 16]), we define a probability space called the sequence space $\Psi_M = (\Omega, \Delta, P_M)$, where $\Omega = \Sigma^\omega$ is the set of all infinite sequences of states, Δ is the Borel field generated by the basic cylindric sets

$$\Delta(w_0, w_1, \dots, w_n) = \{w \in \Omega \mid \text{such that } w \text{ has prefix } w_0, w_1, \dots, w_n\},$$

and P_M is a probability distribution defined by

$$P_M(\Delta(w_0, w_1, \dots, w_n)) =$$

$$P_I(w_0) \cdot P_T(w_0, w_1) \cdot P_T(w_1, w_2) \cdot \dots \cdot P_T(w_{n-1}, w_n).$$

(Δ is the smallest set that contains the basic cylindric sets, and is closed under negation, and countable union.) Consider a language $L \subseteq \Sigma^\omega$, viewed as a specification, that is measurable w.r.t. the sequence space Ψ_M . We say that \mathcal{M} almost surely satisfies L if $P_M(L) = 1$, that is, if “almost” all computations of \mathcal{M} are in L . In this paper we are concerned with the language of an ABW \mathcal{A} , thus we want to determine whether $P_M(L(\mathcal{A})) = 1$. It is shown in [31] that ω -regular languages are measurable.

The following property of Markov chains is called *ergodicity* and is proved in [16]. Let \mathcal{M} be a Markov chain, then a path of \mathcal{M} , with probability one, enters a bottom

strongly connected component (BSCC) K of \mathcal{M} , and contains every finite path in K infinitely often. In other words, let L_e be the set of infinite words of \mathcal{M} such that every word w in L_e has a suffix that is contained in a BSCC K of \mathcal{M} , and contains every finite path in K infinitely often. Then, $P_M(L_e) = 1$.

3 The Full Automaton and The Local Transition System

In this section we capture the behavior of an AWW \mathcal{A} using two nondeterministic systems. First we define the full automaton, which captures the languages of subsets of the states of \mathcal{A} . Then we define the local transition system, which captures the local relations between subsets of states of \mathcal{A} .

3.1 The Full Automaton

Given a AWW $\mathcal{A} = \langle \Sigma, S, \delta, F \rangle$ (we ignore its initial state), we define its dual AWW $\hat{\mathcal{A}} = \langle \Sigma, S, \hat{\delta}, \hat{F} \rangle$, where the Boolean formula $\hat{\delta}(s, \sigma)$ is obtained from $\delta(s, \sigma)$ by replacing every **true** with **false** and vice versa, and every \vee with \wedge and vice versa, in addition we define $\hat{F} = S \setminus F$. It is easy to see that $\hat{\mathcal{A}}$ is an AWW.

Lemma 2. [24] *Let \mathcal{A} be an AWW and $\hat{\mathcal{A}}$ be its dual AWW. For every state s we have that $L(\mathcal{A}^{(s)}) = \Sigma^\omega \setminus L(\hat{\mathcal{A}}^{(s)})$.*

Given an AWW \mathcal{A} and its dual AWW $\hat{\mathcal{A}}$ we define the state space of the full automaton as a subset of $2^S \times 2^S \times 2^S \times 2^S$. We start with the following definition.

Definition 7. *A tuple (Q_1, Q_2, Q_3, Q_4) is consistent if $Q_2 = S \setminus Q_1$, $Q_3 \subseteq Q_1 \setminus F$, and $Q_4 \subseteq Q_2 \setminus \hat{F}$.*

Definition 8. *Given an AWW $\mathcal{A} = \langle \Sigma, S, \delta, F \rangle$ we define its full automaton as the NBW $\mathcal{A}_f = \langle \Sigma, S_f, \delta_f, F_f \rangle$ where*

- S_f is the set of consistent tuples over $2^S \times 2^S \times 2^S \times 2^S$.
- A state (Q'_1, Q'_2, Q'_3, Q'_4) is in $\delta_f((Q_1, Q_2, Q_3, Q_4), \sigma)$ if $Q'_1 \models \wedge_{s \in Q_1} \delta(s, \sigma)$, $Q'_2 \models \wedge_{s \in Q_2} \hat{\delta}(s, \sigma)$, and either:
 1. $Q_3 = Q_4 = \emptyset$, $Q'_3 = Q'_1 \setminus F$, and $Q'_4 = Q'_2 \setminus \hat{F}$
 2. $Q_3 \neq \emptyset$ or $Q_4 \neq \emptyset$, there exists $Y_3 \subseteq Q'_1$ such that $Y_3 \models \wedge_{s \in Q_3} \delta(s, \sigma)$ and $Q'_3 = Y_3 \setminus F$, and there exists $Y_4 \subseteq Q'_2$ such that $Y_4 \models \wedge_{s \in Q_4} \hat{\delta}(s, \sigma)$ and $Q'_4 = Y_4 \setminus \hat{F}$
- $F_f = \{(Q_1, Q_2, Q_3, Q_4) \in S_f | Q_3 = Q_4 = \emptyset\}$

The full automaton can be considered as an intersection of the two NBWs $N(\mathcal{A})$ and $N(\hat{\mathcal{A}})$. The definition of the full automaton restricts the states of the intersection to consistent tuples.

Theorem 2. *Let \mathcal{A} be an AWW and let \mathcal{A}_f be its full automaton, let $Q \subseteq S$ be a set of states, then for every state (Q_1, Q_2, Q_3, Q_4) such that $Q_1 = Q$ we have that*

$$\bigcap_{s \in Q} L(\mathcal{A}^{(s)}) \bigcap_{s \notin Q} \overline{L(\mathcal{A}^{(s)})} = L(\mathcal{A}_f^{(Q_1, Q_2, Q_3, Q_4)})$$

Theorem 2 is proved in Appendix A.1. We now present more properties of the full automaton. We use these properties later.

Definition 9. Let \mathcal{A} be an ABW and let w be an infinite word. We define the type of w w.r.t. \mathcal{A} as the set $\text{type}_{\mathcal{A}}(w) = \{s \mid \mathcal{A}^{(s)} \text{ accepts } w\}$.

The following lemma is a direct consequence of Theorem 2.

Lemma 3. Let \mathcal{A}_f be full automaton, and let (Q_1, Q_2, Q_3, Q_4) and (Q'_1, Q'_2, Q'_3, Q'_4) be states of \mathcal{A}_f . Then,

- If $Q_1 = Q'_1$ then $L(\mathcal{A}_f^{(Q_1, Q_2, Q_3, Q_4)}) = L(\mathcal{A}_f^{(Q'_1, Q'_2, Q'_3, Q'_4)})$.
- If $Q_1 \neq Q'_1$ then $L(\mathcal{A}_f^{(Q_1, Q_2, Q_3, Q_4)}) \cap L(\mathcal{A}_f^{(Q'_1, Q'_2, Q'_3, Q'_4)}) = \emptyset$.

Lemma 3 and Theorem 2 imply that the first element Q_1 of the states of \mathcal{A}_f characterizes a distinct language.

Definition 10. The language of Q_1 is defined as $L(Q_1) = L(\mathcal{A}_f^{(Q_1, S \setminus Q_1, \emptyset, \emptyset)})$.

3.2 The local transition system

As observed above, it is sufficient to look at Q_1 in order to determine the language of a state of \mathcal{A}_f . Recall that $L(Q_1) = L(\mathcal{A}^{(Q_1, S \setminus Q_1, \emptyset, \emptyset)})$. We observe that if there exists a transition $((Q_1, Q_2, Q_3, Q_4), \sigma, (Q'_1, Q'_2, Q'_3, Q'_4))$ in δ_f , then for every state of the form (Q_1, Q_2, Y_3, Y_4) there exists a state (Q'_1, Q'_2, Y'_3, Y'_4) such that the transition $((Q_1, Q_2, Y_3, Y_4), \sigma, (Q'_1, Q'_2, Y'_3, Y'_4))$ is in δ_f .

These observations imply that there are some local relationships between the languages of the states of \mathcal{A}_f . Indeed, if a word w is in $L((Q'_1, Q'_2, Q'_3, Q'_4))$ then for the word $\sigma \cdot w$ that is in $L((Q_1, Q_2, Q_3, Q_4))$, there exists a state of the form (Q'_1, Q'_2, Y'_3, Y'_4) that is in $\delta_f((Q_1, Q_2, Q_3, Q_4), \sigma)$. Thus, we can say that there exists a transition on σ from $L(Q_1)$ to $L(Q'_1)$. The local transition system captures these relationships.

Definition 11. Given an AWW $\mathcal{A} = \langle \Sigma, S, \delta, F \rangle$ we define its local transition system as $T_{\mathcal{A}} = \langle \Sigma, S_T, \delta_T \rangle$ where

- S_T is the set of subsets of S and δ_T is a function from S_T to 2^{S_T} .
- A state Q' is in $\delta_T(Q, \sigma)$ if $Q' \models \wedge_{s \in Q} \delta(s, \sigma)$ and $(S \setminus Q') \models \wedge_{s \notin Q} \hat{\delta}(s, \sigma)$.

Example 2. We now present an example of a full automaton and a local transition system. The example is presented at Figure 1. For simplicity we use a deterministic automaton \mathcal{A} . The figure shows \mathcal{A} 's dual automaton $\hat{\mathcal{A}}$, the full automaton \mathcal{A}_f , and the local transition system $T_{\mathcal{A}}$. Note that $\hat{\mathcal{A}}$ has $\hat{F} = S$, thus for every state (Q_1, Q_2, Q_3, Q_4) of \mathcal{A}_f , we have $Q_4 = \emptyset$. For this reason, and since Q_2 is always equal to $S \setminus Q_1$, we only write the sets Q_1 and Q_3 inside the states. \square

The definitions of the full automaton and the local transition system implies the following lemma:

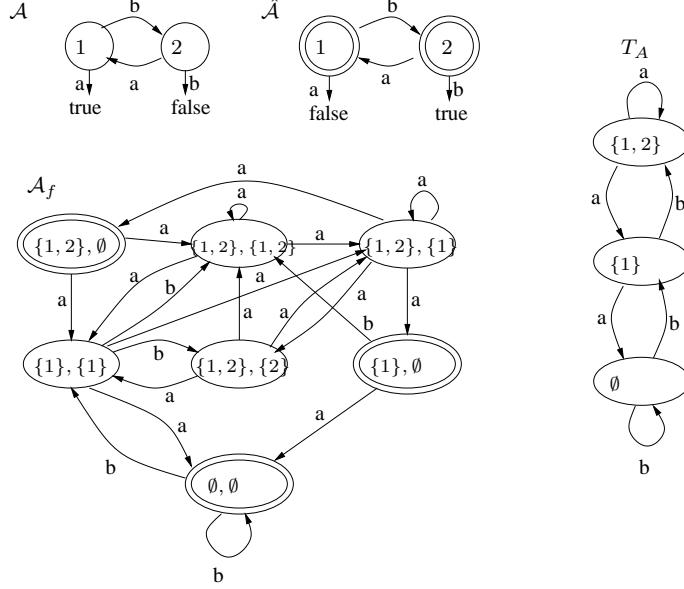


Fig. 1. An example of a full automaton \mathcal{A}_f and a local transition system T_A .

Lemma 4. Let \mathcal{A} be an AWW and let \mathcal{A}_f and $T_{\mathcal{A}}$ be its full automaton and local transition system respectively. Let (Q_1, Q_2, Q_3, Q_4) be a state of \mathcal{A}_f , and let σ be a letter in Σ . Then, for every state Q'_1 we have that Q'_1 is in $\delta_T(Q_1, \sigma)$ iff there exists a state of the form (Q'_1, Q'_2, Q'_3, Q'_4) in $\delta_f((Q_1, Q_2, Q_3, Q_4), \sigma)$.

The proof of Lemma 4 is straightforward from the definitions of \mathcal{A}_f and T_A . In particular for every state (Q_1, Q_2, Q_3, Q_4) and infinite word w we have that $\mathcal{A}^{(Q_1, Q_2, Q_3, Q_4)}$ has a run on w iff $T_A^{(Q_1)}$ has a run on w . However, we do not define accepting conditions for the local transition system. Thus, it is possible that $T_A^{(Q_1)}$ has a run on w , but $\mathcal{A}_f^{(Q_1, Q_2, Q_3, Q_4)}$ does not have an accepting run on w .

Lemma 5. Let T_A be a local transition system, and let Q, Q' and Q'' be states of T_A . Let σ be a letter in Σ . If $Q'' \in \delta_T(Q', \sigma)$ and $Q'' \in \delta_T(Q, \sigma)$, then $Q = Q'$.

Lemma 5 is proved in Appendix A.2. When a transition system satisfies the property shown in Lemma 5, we say that the transition system is *reverse deterministic*.

4 Verifying Markov Chains

In this section we construct a product $G_{\mathcal{M}, \mathcal{A}}$ of the Markov chain \mathcal{M} and the local transition system T_A . We show that the problem of checking whether $P_M(L(\mathcal{A})) = 1$, can be reduced to checking for a state (x, Q) of G whether the probability of $L(Q) \cap x \cdot \Sigma^\omega$ is positive. Then, we show how to use the full automaton to solve this problem.

Definition 12. Let \mathcal{A} be an AWW, T_A be \mathcal{A} 's local transition system, and \mathcal{M} be a Markov chain. We define the graph $G_{\mathcal{M}, \mathcal{A}}$ as having vertex set (x, Q) such that x is a state of \mathcal{M} and Q is a state of T_A . An edge $(x, Q) \rightarrow (x', Q')$ is included in $G_{\mathcal{M}, \mathcal{A}}$ if \mathcal{M} has a transition $x \rightarrow x'$ and (Q, x, Q') is a transition in T_A .

When \mathcal{A} and \mathcal{M} are clear from the context, we write G instead of $G_{\mathcal{M}, \mathcal{A}}$. Lemma 5 implies that for every three states (x, Q) , (x', Q') , and (x'', Q'') , if there is a transition from (x, Q) to (x'', Q'') and there is a transition from (x', Q') to (x'', Q'') , then $x \neq x'$. We say that G is *reverse deterministic*.

Example 3. We present in Figure 2 two Markov chains \mathcal{M}_1 and \mathcal{M}_2 . We assume that the initial probability for each state in the Markov chains is $\frac{1}{2}$. The figure also presents the products G_1 and G_2 of \mathcal{M}_1 and \mathcal{M}_2 respectively, with the local transition system T_A from Example 2. \square

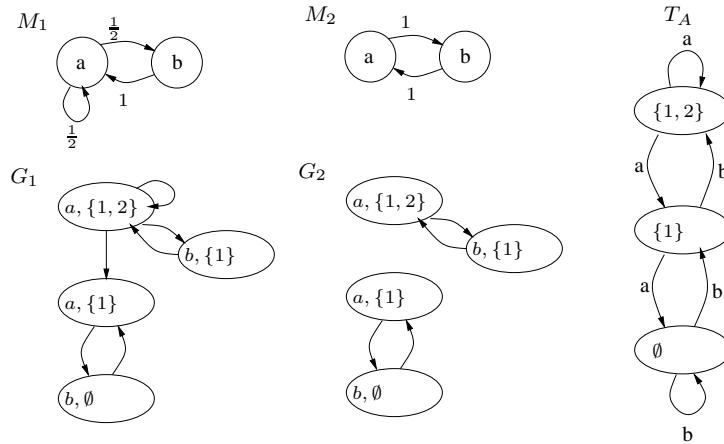


Fig. 2. Two Markov chains \mathcal{M}_1 and \mathcal{M}_2 , and the graphs they impose G_1 and G_2 .

Every infinite path in G projected on the first component gives a path in \mathcal{M} . Conversely, every path of \mathcal{M} is the projection of at least one path in G . In fact, let $w = x_0 \cdot x_1 \dots$ be a path of \mathcal{M} . For each j , let Q_j be the type of the suffix $x_j \cdot x_{j+1} x_{j+2} \dots$. Then for each j there is a transition (Q_j, x_j, Q_{j+1}) in δ_T and thus an edge $(x_j, Q_j) \rightarrow (x_{j+1}, Q_{j+1})$ in G . We call this path the *augmented path* corresponding to w .

Definition 13. For a state (x, Q) of G we denote by $P(x, Q)$ the probability that a path that starts in state x has type Q , namely $P(x, Q) = P_M(\mathcal{M}^{(x)} \text{ has type } Q)$. We call (x, Q) probable if $P(x, Q) > 0$.

The importance of the probable states is demonstrated in the following two lemmas, which are proved in Appendix B.

Lemma 6. $P_M(L(Q)) = \sum_{x \in X} P_I(x) \cdot P(x, Q)$.

Let x be a state of a Markov chain with $P_I(x) > 0$. Then for every state (x, Q) we have that if (x, Q) is probable, then $P_M(L(Q)) > 0$. Thus we conclude Lemma 7.

Lemma 7. *Let s be a state of an AWW \mathcal{A} and let \mathcal{M} be a Markov chain. Then, $P_M(L(s)) < 1$ iff there exists a state x of \mathcal{M} and a set $Q \subseteq S$ such that $P_I(x) > 0$, $s \notin Q$ and the state (x, Q) is probable.*

Thus, in order to determine whether $P_M(L(s_0)) = 1$, it is enough to determine the probable states of G . In the rest of this section we show how to identify the probable states. We define H as the restriction of G to the probable states.

Example 4. Look again at Figure 2. It is easy to see that given that a path of \mathcal{M}_1 starts at state a , the path is of the form $a((ba)+a)^\omega$ with probability 1. $a((ba)+a)^\omega$ is contained in $L(\{1, 2\}) \cup \{(ab)^\omega\}$, since $P_{\mathcal{M}_1}((ab)^\omega) = 0$, we have $P(a, \{1, 2\}) = 1$. Similarly, a path of \mathcal{M}_1 that starts at b is with probability one in $L(\{1\})$, thus, $P(b, \{1\}) = 1$. This implies that in G_1 , H is the subgraph induced by the states $(a, \{1, 2\})$ and $(b, \{1\})$. On the other hand, looking at \mathcal{M}_2 , we see that a path that starts at a is of the form $(ab)^\omega$ and a path that starts at b is of the form $(ba)^\omega$. Thus, in G_2 , H is the subgraph induced by the states $(a, \{1\})$ and (b, \emptyset) . \square

We start with the following observation. Partition the language $L(Q)$ according to the first letter of a word and the type of the suffix that starts from the second letter. Then, for every state (x, Q) of G we have $P(x, Q) = \sum_{(x, Q) \rightarrow (x', Q')} P_T(x, x') \cdot P(x', Q')$. Note that if $(x, Q) \rightarrow (x', Q')$ is an edge of G , then $P_T(x, x') > 0$ and thus $P(x, Q) \geq P_T(x, x') \cdot P(x', Q')$. Hence, if (x', Q') is probable then all its ancestors are probable. This implies that it is sufficient to identify the BSCCs of H and then to construct the set of probable states using backward reachability.

Let C be an SCC of G . If it contains some probable state (x, Q) , then since all the states in C are ancestors of (x, Q) , all states in C are probable. That is, either C is an SCC of H or $C \cap H = \emptyset$. Recall that every path in C projects to a path in \mathcal{M} . So the set of first components of all members of C are in the same SCC, say K , of \mathcal{M} , which is the SCC of \mathcal{M} containing x . We say that C corresponds to K . Note that distinct SCC's of G may correspond to the same K .

Theorem 3 characterizes the SCCs of G which are the BSCCs of H . Before we present the theorem we need the following notation. For a tuple $E = \langle E_1, E_2, \dots, E_n \rangle$, we define $\pi_i(E) = E_i$ to be the i 'th element in E . This notation is extended naturally to sequences of tuples.

Definition 14. *A finite path ρ_G in G is fulfilling if there exists a path ρ_f in \mathcal{A}_f such that $\pi_2(\rho_G) = \pi_1(\rho_f)$, the first state of ρ_f is of the form $(Q_1, Q_2, (Q_1 \setminus F), (Q_2 \setminus \hat{F}))$, and the last state of ρ_f is of the form $(Q'_1, Q'_2, \emptyset, \emptyset)$.*

Theorem 3. *Let C be an SCC of G . Then C is a BSCC of H iff it satisfies the following conditions:*

1. *C corresponds to a BSCC K of \mathcal{M} .*
2. *Every finite path of K is a projection of a path in C .*
3. *C contains a fulfilling path.*

The proof of Theorem 3 is highly nontrivial and is presented at Appendix B.1.

5 Algorithms

In Figure 3 we present the algorithm that determines for an AWW \mathcal{A} and a Markov chain \mathcal{M} whether $P_{\mathcal{M}}(L(\mathcal{A})) = 1$. An extension for exact probability is presented in Appendix D. Theorem 3 implies that the algorithm marks the BSCCs of H . Thus, B is the set of probable states. Lemma 7 implies that the algorithm returns **true** iff $P_{\mathcal{M}}(L(\mathcal{A})) = 1$. Finding SCCs in G that correspond to BSCCs in \mathcal{M} , and doing

```

Inputs: Markov chain  $\mathcal{M} = \langle X, P_I, P_T \rangle$ , AWW  $\mathcal{A} = \langle \Sigma, S, s_0, \delta, F \rangle$ .
Construct the full automaton  $\mathcal{A}_f$  of  $\mathcal{A}$ .
Construct the local transition system  $T_A$  and the graph  $G$ .
Mark all SCCs  $C$  of  $G$  that satisfy:
  1.  $C$  corresponds to a BSCC  $K$  of  $\mathcal{M}$ .
  2. Every finite path of  $K$  is a projection of a path in  $C$ .
  3.  $C$  contains a fulfilling path.
Construct the set  $B$  of all states of  $G$  from which the marked
SCCCs are reachable.
return true iff for every state  $(x, Q) \in B$ , if  $P_I(x) > 0$ , then  $s_0 \in Q$ .

```

Fig. 3. The model-checking algorithm

backward reachability can be done in time linear in the size of G . The most complex part of the algorithm is to identify, SCCs C of G that satisfy:

1. C corresponds to a BSCC K of \mathcal{M} .
2. Every finite path of K is a projection of a path in C .
3. C contains a fulfilling path.

The following lemma is proved in [10]. The only property of G that they use is that G is reverse deterministic.

Lemma 8. *Let C be an SCC of G that corresponds to an SCC K of \mathcal{M} . Then the following are equivalent:*

1. *Every finite path in K is a projection of a path in C .*
2. *No other SCC of G corresponding to K is an ancestor of C .*

Lemma 8 implies that the second task is equivalent to checking whether there is no ancestor SCC of C that corresponds to K . This check can be easily done while scanning the SCCs of G .

Example 5. In G_1 at Figure 2 there are two SCC's that correspond to the single BSCC of \mathcal{M}_1 . The SCC of $(a, \{1, 2\})$ and $(b, \{1\})$ does not have ancestors and contains the fulfilling path $(a, \{1, 2\}), (a, \{1, 2\}), (a, \{1, 2\})$ that corresponds to the path $(\{1, 2\}, \{1, 2\}), (\{1, 2\}, \{1\}), (\{1, 2\}, \emptyset)$ in \mathcal{A}_f , thus, this SCC is the BSCC of H . In G_2 there are two SCCs that correspond to the single BSCC of \mathcal{M}_2 and neither of them have an ancestor. However, only the SCC of $(a, \{1\})$ and (b, \emptyset) has a fulfilling path, thus it is the BSCC of H . \square

We now explain how to check whether an SCC C of G contains a fulfilling path. We construct the product $G_f = \mathcal{M} \times \mathcal{A}_f$, similarly to the construction of G .

Definition 15. Let \mathcal{A} be an AWW, \mathcal{A}_f be \mathcal{A} 's full automaton, and \mathcal{M} be a Markov chain. We define the full graph G_f as having vertex set $(x, (Q_1, Q_2, Q_3, Q_4))$ such that x is a state of \mathcal{M} and (Q_1, Q_2, Q_3, Q_4) is a state of \mathcal{A}_f . An edge $(x, (Q_1, Q_2, Q_3, Q_4)) \rightarrow (x', (Q'_1, Q'_2, Q'_3, Q'_4))$ is included in G_f if \mathcal{M} has a transition $x \rightarrow x'$ and (Q'_1, Q'_2, Q'_3, Q'_4) is in $\delta_f((Q_1, Q_2, Q_3, Q_4), x)$.

Lemma 9. An SCC C of G contains a fulfilling path iff there exists a path $(x_0, (Q_1^0, Q_2^0, Q_3^0, Q_4^0)), (x_1, (Q_1^1, Q_2^1, Q_3^1, Q_4^1)), \dots, (x_n, (Q_1^n, Q_2^n, \emptyset, \emptyset))$ in G_f such that the path $(x_0, Q_1^0), (x_1, Q_1^1), \dots, (x_n, Q_1^n)$ is contained in C , $Q_3^0 = Q_1^0 \setminus F$, and $Q_4^0 = Q_2^0 \setminus \hat{F}$.

Lemma 9 is proved in Appendix C.

Complexity Finding SCCs in G that correspond to BSCCs in \mathcal{M} , and doing backward reachability can be done in linear time and polylogarithmic space in $|G|$. Constructing BSCCs of \mathcal{M} can be done in time linear in $|\mathcal{M}|$, identifying SCCs of G that correspond to these BSCC can be done in time linear in $|G|$. Marking SCCs that do not have ancestors that correspond to the same BSCC in \mathcal{M} can also be done in time linear in $|G|$. Checking that an SCC of G contains a fulfilling path can be done in time linear in $|G_f|$, simply by scanning G_f and G in parallel, thus, the algorithm can be implemented in time linear in $|\mathcal{M} \times \mathcal{A}_f|$. Since the size of \mathcal{A}_f is $2^{O(|\mathcal{A}|)}$, we have that the time complexity of the algorithm is $|\mathcal{M}| \cdot 2^{O(|\mathcal{A}|)}$.

As for space complexity we show that algorithm works in space polynomial in $|\mathcal{A}|$ and polylogarithmic in $|\mathcal{M}|$. We rewrite the conditions of Theorem 3, Lemma 7, and Lemma 8 as follows: $P_M(L(\mathcal{A})) < 1$ iff there exists a probable state (x_0, Q_0) such that $s_0 \notin Q$ and $P_I(x_0) > 0$. This is true iff (x_0, Q_0) reaches a state (x, Q) that is in a BSCC of H , $s_0 \notin Q_0$, and $P_I(x_0) > 0$. That is

1. (x, Q) is reachable from a state (x_0, Q_0) such that $P_I(x_0) > 0$ and $s_0 \notin Q_0$.
2. x is in a BSCC of \mathcal{M} (Theorem 3, (1)). This condition is equivalent to the following: for every state x' of \mathcal{M} we have that if there exists a path in \mathcal{M} from x to x' then there exists a path from x' to x .
3. No other SCC of G that corresponds to the SCC of x in \mathcal{M} is the ancestor of the SCC of (x, Q) (Lemma 8). This condition is equivalent to the following: for every state (x', Q') , if there exists a path from (x', Q') to (x, Q) , then either there exists a path from (x, Q) to (x', Q') , or there is no path from x to x' .
4. The SCC of (x, Q) contains a fulfilling path (Theorem 3, (3)). By Lemma 9 this condition is equivalent to the following: there exists a path in G_f from a state $(x', (Q'_1, Q'_2, Q'_1 \setminus F, Q'_2 \setminus \hat{F}))$ to a state $(x'', (Q''_1, Q''_2, \emptyset, \emptyset))$ such that the projection of the path on G is contained in the SCC of (x, Q) . This condition is equivalent to: there is a path from a state $(x', (Q'_1, Q'_2, Q'_1 \setminus F, Q'_2 \setminus \hat{F}))$ to a state $(x'', (Q''_1, Q''_2, \emptyset, \emptyset))$ in G_f and there are paths from (x, Q) to (x'', Q''_1) , from (x'', Q''_1) to (x', Q') , and from $((x', Q'))$ to (x, Q) in G .

In [27] it is shown that checking whether there is a path from one state to another in a graph with n states requires $\log^2(n)$ space. This implies that the conditions above can be checked in space $O(\log^2(|G_f|)) = \log^2(|\mathcal{M}| \cdot 2^{O(|\mathcal{A}|)}) = O(\log^2(|\mathcal{M}|) + \log(|\mathcal{M}|) \cdot |\mathcal{A}| + |\mathcal{A}|^2) = O(\log^2(|\mathcal{M}|) + |\mathcal{A}|^2)$.

6 Concluding remarks

We presented here an optimal solution to the general problem of linear-time probabilistic model checking with respect to ω -regular specifications, expressed by alternating automata. Beyond the interest in the problem itself, our solution is interesting from a theoretical perspective, since the concept of full automaton may have other applications. More work is needed in reducing our result to practice. One direction is to extend the *ProbaTaf* system, which currently handles LTL specifications of Markov chain [11], to alternating automata specifications. Another, is to combine the symbolic approach to alternating automata [21] with the symbolic approach to probabilistic model checking [3].

References

1. R. Armoni, D. Bustan, O. Kupferman, and M. Y. Vardi. Resets vs. aborts in linear temporal logic. In *Int'l Conf. on Tools and Algorithms for Construction and Analysis of Systems*, pages 65–80, 2003.
2. R. Armoni, L. Fix, R. Gerth, B. Ginsburg, T. Kanza, A. Landver, S. Mador-Haim, A. Tiemeyer, E. Singerman, M.Y. Vardi, and Y. Zbar. The ForSpec temporal language: A new temporal property-specification language. In *Proc. 8th Int'l Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'02)*, LNCS 2280, pages 296–311, 2002.
3. C. Baier, E.M. Clarke, V. Hartonas-Garmhausen, M.Z. Kwiatkowska, and M. Ryan. Symbolic model checking for probabilistic processes. In *Automata, Languages and Programming, 24th Int'l Colloq.*, LNCS 1256, pages 430–440, 1997.
4. I. Beer, S. Ben-David, C. Eisner, D. Fisman, A. Gringauze, and Y. Rodeh. The temporal logic sugar. In *Proc. Conf. on Computer-Aided Verification*, LNCS 2102, pages 363–367, 2001.
5. P. Berman and J.A Garay. Randomized distributed agreement revisited. In *Proceedings of the 23rd Int'l Symp. on Fault-Tolerant Computing (FTCS '93)*, pages 412–421, 1993.
6. E.M. Clarke, E.A. Emerson, and A.P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Trans. on Programming Languages and Systems*, 8(2):244–263, 1986.
7. E.M. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 1999.
8. R. Cole, B.M. Maggs, F. M. auf der Heide, A. Richa M. Mitzenmacher, K. Schroder, R. Sitaraman, and B. Vocking. Randomized protocols for low-congestion circuit routing in multistage interconnection networks. In *30th ACM Symp. on Theo. of Comp. (STOC)*, pages 378–388, 1998.
9. C. Courcoubetis and M. Yannakakis. Markov decision processes and regular events. In *Proc. 17th Int'l Coll. on Automata Languages and Programming*, volume 443, pages 336–349. LNCS, 1990.

10. C. Courcoubetis and M. Yannakakis. The complexity of probabilistic verification. *Journal of the ACM*, 42(4):857–907, 1995.
11. J. M. Couvreur, N. Saheb, and G. Sutre. An optimal automata approach to LTL model checking of probabilistic systems. In *Proc. 10th Int. Conf. Logic for Programming, Artificial Intelligence, and Reasoning (LPAR'2003), Almaty, Kazakhstan, Sep. 2003*, volume 2850 of *Lecture Notes in Artificial Intelligence*. Springer, 2003.
12. C. Fritz and T. Wilke. State space reductions for alternating Büchi automata: Quotienting by simulation equivalences. In *FST TCS 2002: Foundations of Software Technology and Theoretical Computer Science: 22nd Conf.*, volume 2556 of *LNCS*, pages 157–168, 2002.
13. P. Gastin and D. Oddoux. Fast LTL to Büchi automata translation. In *Computer Aided Verification, Proc. 13th Int'l Conf.*, volume 2102 of *LNCS*, pages 53–65, 2001.
14. G.J. Holzmann. The model checker SPIN. *IEEE Trans. on Software Engineering*, 23(5):279–295, May 1997. Special issue on Formal Methods in Software Practice.
15. J.G. Kemeny and J.L. Snell. *Finite Markov Chains*. Van Nostrand, Princeton, 1960.
16. J.G. Kemeny, J.L. Snell, and A.W. Knapp. *Denumerable Markov Chains*. Springer-Verlag, 1976.
17. O. Kupferman and M.Y. Vardi. Weak alternating automata are not that weak. In *Proc. 5th Israeli Symp. on Theory of Computing and Systems*, pages 147–158. IEEE Computer Society Press, 1997.
18. R.P. Kurshan. *FormalCheck User's Manual*. Cadence Design, Inc., 1998.
19. O. Lichtenstein and A. Pnueli. Checking that finite state concurrent programs satisfy their linear specification. In *Proc. 12th ACM Symp. on Principles of Programming Languages*, pages 97–107, 1985.
20. O. Lichtenstein, A. Pnueli, and L. Zuck. The glory of the past. In *Logics of Programs*, volume 193 of *Lecture Notes in Computer Science*, pages 196–218, Brooklyn, June 1985. Springer-Verlag.
21. S. Merz. Weak alternating automata in Isabelle/HOL. In J. Harrison and M. Aagaard, editors, *Theorem Proving in Higher Order Logics: 13th International Conference*, volume 1869 of *Lecture Notes in Computer Science*, pages 423–440. Springer-Verlag, 2000.
22. S. Miyano and T. Hayashi. Alternating finite automata on ω -words. *Theoretical Computer Science*, 32:321–330, 1984.
23. M.J. Morley. Semantics of temporal e . In T. F. Melham and F.G. Moller, editors, *Banff'99 Higher Order Workshop (Formal Methods in Computation)*. University of Glasgow, Department of Computing Science Technical Report, 1999.
24. D.E. Muller, A. Saoudi, and P.E. Schupp. Alternating automata, the weak monadic theory of the tree and its complexity. In *Proc. 13th Intel Colloq. on Automata, Languages and Programming*, volume 226 of *LNCS*, 1986.
25. J.P. Queille and J. Sifakis. Specification and verification of concurrent systems in Cesar. In *Proc. 5th Int'l Symp. on Programming*, volume 137 of *LNCS*, pages 337–351, 1981.
26. S. Safra. On the complexity of ω -automata. In *Proc. 29th IEEE Symp. on Foundations of Computer Science*, pages 319–327, White Plains, October 1988.
27. W.J. Savitch. Relationship between nondeterministic and deterministic tape complexities. *Journal on Computer and System Sciences*, 4:177–192, 1970.
28. A.P. Sistla and E.M. Clarke. The complexity of propositional linear temporal logic. *J. ACM*, 32:733–749, 1985.
29. S. Thrun. Probabilistic algorithms in robotics. *AI Magazine*, 21(4):93–109, 2000.
30. M. Y. Vardi. Probabilistic linear-time model checking: An overview of the automata-theoretic approach. In *Formal Methods for Real-Time and Probabilistic Systems: 5th Int'l AMAST Workshop*, volume 1601 of *LNCS*, pages 265–276, 1999.

31. M.Y. Vardi. Automatic verification of probabilistic concurrent finite-state programs. In *Proc. 26th IEEE Symp. on Foundations of Computer Science*, pages 327–338, Portland, October 1985.
32. M.Y. Vardi. A temporal fixpoint calculus. In *Proc. 15th ACM Symp. on Principles of Programming Languages*, pages 250–259, San Diego, January 1988.
33. M.Y. Vardi. Nontraditional applications of automata theory. In *Proc. Inte'l Symp. on Theoretical Aspects of Computer Software*, volume 789, pages 575–597. LNCS, 1994.
34. M.Y. Vardi. An automata-theoretic approach to linear temporal logic. In *Logics for Concurrency: Structure versus Automata*, volume 1043 of *LNCS*, pages 238–266, 1996.
35. M.Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *Proc. 1st Symp. on Logic in Computer Science*, pages 332–344, Cambridge, June 1986.
36. M.Y. Vardi and P. Wolper. Reasoning about infinite computations. *Information and Computation*, 115(1):1–37, November 1994.
37. P. Wolper. Temporal logic can be more expressive. *Information and Control*, 56(1–2):72–99, 1983.

A Proofs for Section 3

A.1 Proof of Theorem 2

Lemma 10. Let \mathcal{A} be an AWW, let s be a state in S , and let σ be a letter in Σ . Then, for every $Q \subseteq S$, we have that $Q \models \delta(s, \sigma)$ iff $S \setminus Q \not\models \hat{\delta}(s, \sigma)$.

Lemma 10 can be proved by induction on the structure of the formula $\delta(s, \sigma)$.

Lemma 11. Let \mathcal{A} be an AWW, let w be an infinite word, and let Q be a subset of S s.t w is in $\bigcap_{s \in Q} L(\mathcal{A}^{(s)}) \bigcap_{s \notin Q} \overline{L(\mathcal{A}^{(s)})}$. Then, $Q = \text{type}_{\mathcal{A}}(w)$.

Proof. Since $w \in \bigcap_{s \in Q} L(\mathcal{A}^{(s)})$, we have $Q \subseteq \text{type}_{\mathcal{A}}(w)$. Since $w \in \bigcap_{s \notin Q} \overline{L(\mathcal{A}^{(s)})}$, we have $\text{type}_{\mathcal{A}}(w) \subseteq Q$, thus, $Q = \text{type}_{\mathcal{A}}(w)$. \square

Lemma 12. Let \mathcal{A} be an AWW, let $\hat{\mathcal{A}}$ be its dual, and let w be an infinite word. Then $S \setminus \text{type}_{\mathcal{A}}(w) = \text{type}_{\hat{\mathcal{A}}}(w)$.

Proof. Let s be a state, then $s \in S \setminus \text{type}_{\mathcal{A}}(w)$ iff $w \notin L(\mathcal{A}^{(s)})$. By Lemma 2, $w \notin L(\mathcal{A}^{(s)})$ iff $w \in L(\hat{\mathcal{A}}^{(s)})$ iff $s \in \text{type}_{\hat{\mathcal{A}}}(w)$. \square

Lemma 13. Let \mathcal{A} be an AWW and let w be an infinite word. Then for every i and every $s \in \text{type}_{\mathcal{A}}(w^i)$, we have $\text{type}_{\mathcal{A}}(w^{i+1}) \models \delta(s, w_i)$

Proof. Let s be a state in $\text{type}_{\mathcal{A}}(w^i)$, then $\mathcal{A}^{(s)}$ accepts w^i . Let Q' be the set of successors of s in an accepting run tree of $\mathcal{A}^{(s)}$ on w^i . Then, for every $t \in Q'$ we have that $\mathcal{A}^{(t)}$ accepts w^{i+1} , thus $Q' \subseteq \text{type}_{\mathcal{A}}(w^{i+1})$. Since, $Q' \models \delta(s, w_i)$, we have $\text{type}_{\mathcal{A}}(w^{i+1}) \models \delta(s, w_i)$. \square

Definition 16. Let \mathcal{A} be an AWW and \mathcal{A}_f be the full automaton of \mathcal{A} . Let w be an infinite word. We define $\text{Max}_{\mathcal{A}}(w)$ to be the state $(\text{type}_{\mathcal{A}}(w), \text{type}_{\hat{\mathcal{A}}}(w), \text{type}_{\mathcal{A}}(w) \setminus F, \text{type}_{\hat{\mathcal{A}}}(w) \setminus \hat{F})$ of \mathcal{A}_f .

Lemma 12 implies that $\text{Max}_{\mathcal{A}}(w)$ is a consistent tuple.

Lemma 14. Let w be an infinite word and let \mathcal{A} be an AWW. Then there exists a run of \mathcal{A}_f on a prefix of w that starts at $\text{Max}_{\mathcal{A}}(w)$ and that reaches a state in F_f .

Proof. Let $(Q_1^0, Q_2^0, Q_3^0, Q_4^0) = \text{Max}_{\mathcal{A}}(w)$. We define two sets of trees. Φ_3 is a set of $|Q_1^0|$ many run trees s.t. for every $s \in Q_1^0$ there exists a unique tree in Φ_3 that is an accepting run tree of $\mathcal{A}^{(s)}$ on w . Similarly, we define Φ_4 as a set of $|Q_2^0|$ many run trees s.t. for every $t \notin Q_1^0$ there exists a unique tree in Φ_4 that is an accepting run tree of $\hat{\mathcal{A}}^{(t)}$ on w . We define $\Phi = \Phi_3 \cup \Phi_4$. Note that every path of a tree in Φ_3 contains infinitely many accepting states in F , and every path of a tree in Φ_4 contains infinitely many accepting states in \hat{F} . Furthermore, the branching degree of every node in these tree is bounded by $|S|$. By König's lemma, there exists an index i s.t. for every tree in Φ and every path ξ in the tree there exists $k \leq i$ s.t. ξ_k is an accepting state.

We construct a run of $\mathcal{A}_f^{(Q_1^0, Q_2^0, Q_3^0, Q_4^0)}$ on w inductively, until it reaches a state in F_f . The run satisfies the following properties:

1. For every $k \leq i$ we have that if a state s is in Q_3^k , then there exists a path ξ in a tree in Φ_3 s.t. $\xi_0, \xi_1, \dots, \xi_k$ does not contain states in F and $\xi_k = s$.
2. For every $k \leq i$ we have that if a state t is in Q_4^k , then there exists a path ξ in a tree in Φ_4 s.t. $\xi_0, \xi_1, \dots, \xi_k$ does not contain states in \hat{F} and $\xi_k = t$.

The first state of the run is $(Q_1^0, Q_2^0, Q_3^0, Q_4^0)$. It is easy to see that the properties above hold for $k = 0$. Suppose that for some $k < i$, we have a prefix of a run of \mathcal{A}_f that satisfies the properties above. We define $Q_1^{k+1} = \text{type}_{\mathcal{A}}(w^{k+1})$, $Q_2^{k+1} = S \setminus Q_1^{k+1}$. As for Q_3^{k+1} and Q_4^{k+1} , the induction hypothesis implies that for every state s in Q_3^k there exists a path ξ^s in a tree in Φ_3 s.t. $\xi_0^s, \xi_1^s, \dots, \xi_k^s$ does not contain states in F and $\xi_k^s = s$. Let Y_3^s be the set of successors of ξ_k^s in the tree. We define $Y_3 = \bigcup_{s \in Q_3^k} Y_3^s$, and $Q_3^{k+1} = Y_3 \setminus F$. Since for every s in Q_3^k we have that the successors of ξ_k^s in the run tree accepts w^{k+1} , we have that $Y_3 \subseteq \text{type}_{\mathcal{A}}(w^{k+1})$, thus, $Y_3 \subseteq Q_1^{k+1}$. In a similar way we define $Y_4 \subseteq Q_2^{k+1}$ using trees of Φ_4 , and define $Q_4^{k+1} = Y_4 \setminus \hat{F}$. Since $Y_3 \subseteq Q_1^{k+1}$ and $Y_4 \subseteq Q_2^{k+1}$, the state $(Q_1^{k+1}, Q_2^{k+1}, Q_3^{k+1}, Q_4^{k+1})$ is consistent.

Obviously, the properties above holds for $k+1$. We prove that $(Q_1^{k+1}, Q_2^{k+1}, Q_3^{k+1}, Q_4^{k+1})$ is in $\delta_f((Q_1^k, Q_2^k, Q_3^k, Q_4^k), w_k)$. Lemma 13 implies that for every state $s \in Q_1^k$ we have $Q_1^{k+1} \models \delta(s, w_k)$. Lemma 12 implies that $Q_2^{k+1} = \text{type}_{\mathcal{A}}(w^{k+1})$, Lemma 13 implies that for every $t \in Q_2^k$, we have $Q_2^{k+1} \models \hat{\delta}(t, w_k)$. The definition of Y_3 implies that for every state s in Q_3^k we have $Y_3 \models \delta(s, w_k)$. Similarly, for every $t \in Q_4^k$ we have $Y_4 \models \hat{\delta}(s, w_k)$. This completes the proof that $(Q_1^{k+1}, Q_2^{k+1}, Q_3^{k+1}, Q_4^{k+1})$ is in $\delta_f((Q_1^k, Q_2^k, Q_3^k, Q_4^k), w_k)$.

Since every path in a tree in Φ contains an accepting state before the i 'th state, the properties above imply that the run reaches an accepting state before the i 'th state. \square

Lemma 15. *Let \mathcal{A} be a AWW and let \mathcal{A}_f be its full automaton, let $Q \subseteq S$ be a set of states, then for every state (Q_1, Q_2, Q_3, Q_4) s.t. $Q_1 = Q$ we have that*

$$\bigcap_{s \in Q} L(\mathcal{A}^{(s)}) \bigcap_{s \notin Q} \overline{L(\mathcal{A}^{(s)})} \subseteq L(\mathcal{A}_f^{(Q_1, Q_2, Q_3, Q_4)})$$

Proof. We prove that every w in $\bigcap_{s \in Q} L(\mathcal{A}^{(s)}) \bigcap_{s \notin Q} \overline{L(\mathcal{A}^{(s)})}$ is also in $L(\mathcal{A}_f^{(Q_1, Q_2, Q_3, Q_4)})$. We construct the run of \mathcal{A}_f inductively in finite sequences, s.t. for every sequence, the last state $(Q_1^i, Q_2^i, Q_3^i, Q_4^i)$ in the sequence is $\text{Max}_{\mathcal{A}}(w^i)$. Moreover, every sequence except for the first, contains an accepting state. Since, the run contains infinitely many sequences, the run is accepting.

- Base: Lemma 11 implies that $Q_1 = \text{type}_{\mathcal{A}}(w)$. Lemma 19 implies that there exists a transition from (Q_1, Q_2, Q_3, Q_4) to $\text{Max}_{\mathcal{A}}(w^1)$ on w_0 . This is the first sequence.
- Induction step: Suppose that the i 'th sequence ends at state $(Q_1^k, Q_2^k, Q_3^k, Q_4^k)$ s.t. $(Q_1^k, Q_2^k, Q_3^k, Q_4^k) = \text{Max}_{\mathcal{A}}(w^k)$. Then, Lemma 14 implies that there exists a run of \mathcal{A}_f that starts from $(Q_1^k, Q_2^k, Q_3^k, Q_4^k)$ and reaches an accepting state $(Q_1^j, Q_2^j, \emptyset, \emptyset)$ for some $j > k$. The definition of \mathcal{A}_f implies that there exists a transition from $(Q_1^j, Q_2^j, \emptyset, \emptyset)$ to $\text{Max}_{\mathcal{A}}(w^{j+1})$ on w_j . We take $(Q_1^{k+1}, Q_2^{k+1}, Q_3^{k+1}, Q_4^{k+1}), \dots, (Q_1^j, Q_2^j, \emptyset, \emptyset), \text{Max}_{\mathcal{A}}(w^{j+1})$ to be the $i+1$ 'th sequence. \square

Lemma 16. Let \mathcal{A} be a AWW and let \mathcal{A}_f be its full automaton, let $Q \subseteq S$ be a set of states, then for every state (Q_1, Q_2, Q_3, Q_4) s.t. $Q_1 = Q$ we have that $\bigcap_{s \in Q} L(\mathcal{A}^{(s)}) \bigcap_{s \notin Q} \overline{L(\mathcal{A}^{(s)})} \supseteq L(\mathcal{A}_f^{(Q_1, Q_2, Q_3, Q_4)})$

Proof. We prove that every w in $L(\mathcal{A}_f^{(Q_1, Q_2, Q_3, Q_4)})$ is also in $\bigcap_{s \in Q} L(\mathcal{A}^{(s)}) \bigcap_{s \notin Q} \overline{L(\mathcal{A}^{(s)})}$. Let $\rho_f = (Q_1^0, Q_2^0, Q_3^0, Q_4^0), (Q_1^1, Q_2^1, Q_3^1, Q_4^1), \dots$ be an accepting run of $\mathcal{A}_f^{(Q_1, Q_2, Q_3, Q_4)}$ on w and let s be a state in Q_1^0 . We need to show that w is in $\mathcal{A}^{(s)}$. Let $\mathcal{A}_n = N(\mathcal{A})$ as defined in Definition 5. We prove that $\mathcal{A}_n^{(Q_1, Q_3)}$ has an accepting run on w , by Theorem 1 this implies that w is in $L(\mathcal{A}^{(s)})$. Similarly, we can prove that w is in $L(\hat{\mathcal{A}}^{(t)})$ for every state t in Q_2^0 .

Let $i_0 = -1$ and let i_1, i_2, \dots be an infinite sequence of indices s.t. for every $j \geq 1$, the i_j 'th state of ρ is in F_f . We construct a run $\rho_n = (Y_1^0, Y_2^0), (Y_1^1, Y_2^1), \dots$ s.t. For every j and $i_j < k \leq i_{j+1}$ we define $Y_1^k = Q_1^k$, and

$$Y_2^k = \begin{cases} Y_1^k \setminus F & \text{if } Y_2^l = \emptyset \text{ for some } i_j < l < k \\ Q_3^k & \text{otherwise} \end{cases}$$

We prove that ρ_n is an accepting run of \mathcal{A}_n on w . We prove by induction on j that for every $j \geq 0$ we have that

1. $Y_1^{i_j+1} = Q_1^{i_j+1}$ and $Y_2^{i_j+1} = Q_3^{i_j+1}$.
2. For every $i_j < k \leq i_{j+1}$, there exists a transition from (Y_1^k, Y_2^k) to (Y_1^{k+1}, Y_2^{k+1}) on w_k .
3. There exists $i_j < k \leq i_{j+1}$ s.t. $Y_2^k = \emptyset$

Since for every $j \geq 0$ there exists $k > i_j$ s.t. $Y_2^k = \emptyset$, the run is accepting.

We start with two observations. The definitions of \mathcal{A}_f and \mathcal{A}_n imply the following observation.

Observation 17 For a state (Q_1, Q_2, Q_3, Q_4) of \mathcal{A}_f s.t. $Q_3 \neq \emptyset$, a transition from (Q_1, Q_2, Q_3, Q_4) to (Q'_1, Q'_2, Q'_3, Q'_4) on σ in \mathcal{A}_f implies a transition from (Q_1, Q_3) to (Q'_1, Q'_3) in \mathcal{A}_n .

Recall that if there exists a transition in \mathcal{A}_f from (Q_1, Q_2, Q_3, Q_4) to (Q'_1, Q'_2, Q'_3, Q'_4) on σ , then there exists a transition in \mathcal{A}_f from (Q_1, Q_2, Q_3, Q_4) to $(Q'_1, Q'_2, Q'_1 \setminus F, Q'_2 \setminus \bar{F})$ on σ . This implies that:

Observation 18 If there exists a transition in \mathcal{A}_f from (Q_1, Q_2, Q_3, Q_4) to (Q'_1, Q'_2, Q'_3, Q'_4) on σ , then there exist a transition in \mathcal{A}_n from (Q_1, Q_3) to $(Q'_1, Q'_1 \setminus F)$ on σ .

Next, we prove the induction claim.

- Base case $j = 0$: The definition of ρ_n implies property 1. Let l be the smallest index s.t. $i_j < l \leq i_{j+1}$ and $Q_3^l = \emptyset$. Then Observation 17 implies that for every $i_j < k < l$ property 2 holds. By the definition of \mathcal{A}_n , property 3 holds in index $k = l$. Observation 18 implies that property 2 holds for $l \leq k \leq i_{j+1}$.

- Induction step: Assume that the three properties hold for j , we prove that they hold for $j + 1$. Since $(Q_1^{i_j}, Q_2^{i_j}, Q_3^{i_j}, Q_4^{i_j})$ is in F_f , we have $Q_3^{i_j+1} = Q_1^{i_j+1} \setminus F$. The induction hypothesis implies that for some $i_{j-1} < k \leq i_j$, we have $Y_2^k = \emptyset$ thus $Y_2^{i_j+1} = Y_1^{i_j+1} \setminus F$, thus property 1 holds. Properties 2 and 3 can be proved as in the base case. \square

Lemma 16 completes the proof of Theorem 2.

A.2 Proofs for the rest of Section 3

Lemma 19. *Let \mathcal{A} be an AWW, let \mathcal{A}_f be its full automaton, and let w be an infinite word. Then, there exist a transition in \mathcal{A}_f from every state of the form $(\text{type}_{\mathcal{A}}(w), S \setminus \text{type}_{\mathcal{A}}(w), Q_3, Q_4)$ to $\text{Max}_{\mathcal{A}}(w^1)$ on w_0 .*

Proof. By Lemma 12, we have $S \setminus \text{type}_{\mathcal{A}}(w) = \text{type}_{\hat{\mathcal{A}}}(w)$. Lemma 13 implies that for every state s in $\text{type}_{\mathcal{A}}(w)$, we have $\text{type}_{\mathcal{A}}(w^1) \models \delta(s, w_0)$, and that for every state t in $\text{type}_{\hat{\mathcal{A}}}(w)$, we have $\text{type}_{\hat{\mathcal{A}}}(w^1) \models \hat{\delta}(t, w_0)$. If $Q_3 = Q_4 = \emptyset$, then the definition of δ_f implies the lemma directly. Otherwise, since $(\text{type}_{\mathcal{A}}(w), S \setminus \text{type}_{\mathcal{A}}(w), Q_3, Q_4)$ is consistent, we have that $Q_3 \subseteq \text{type}_{\mathcal{A}}(w) \setminus F$, and $Q_4 \subseteq \text{type}_{\hat{\mathcal{A}}}(w) \setminus \hat{F}$. We take $Y_3 = \text{type}_{\mathcal{A}}(w^1)$ and $Y_4 = \text{type}_{\hat{\mathcal{A}}}(w^1)$. These sets satisfy the conditions of δ_f . \square

Theorem 2 implies the following lemma.

Lemma 20. *For an AWW \mathcal{A} and set of state Q , we have $L(Q) = \{w | \text{type}_{\mathcal{A}}(w) = Q\}$.*

Lemma 5 Let $T_{\mathcal{A}}$ be a local transition system, and let Q, Q' and Q'' be states of $T_{\mathcal{A}}$. Let σ be a letter in Σ . If $Q'' \in \delta_T(Q', \sigma)$ and $Q'' \in \delta_T(Q, \sigma)$, then $Q = Q'$.

Proof. Assume to the contrary that the lemma does not hold, meaning, $Q'' \in \delta_T(Q', \sigma)$ and $Q'' \in \delta_T(Q, \sigma)$, but $Q \neq Q'$. Then, w.l.o.g. there exists a state s in Q that is not in Q' . The definition of $T_{\mathcal{A}}$ implies that $Q'' \models \delta(s, \sigma)$. Lemma 10 implies that $S \setminus Q'' \not\models \hat{\delta}(s, \sigma)$. Thus, $Q'' \notin \delta_T(Q', \sigma)$, contradiction. \square

B Proofs for Section 4

Lemma 6 $P_M(L(Q)) = \sum_{x \in X} P_I(x) \cdot P(x, Q)$.

Proof. We partition the language $L(Q)$ according to the first letter of the word. Then, we have that $P_M(L(Q) \cap \{w | w \text{ starts with } x\}) = P_I(x) \cdot P(x, Q)$. Thus, $P_M(L(Q)) = \sum_{x \in X} P_I(x) \cdot P(x, Q)$. \square

Let x be a state of a Markov chain with $P_I(x) > 0$. Then for every state (x, Q) we have that if (x, Q) is probable, then $P_M(L(Q)) > 0$. Thus we conclude the following lemma.

Lemma 7 Let s be a state of an AWW \mathcal{A} and let \mathcal{M} be a Markov chain. Then, $P_{\mathcal{M}}(L(s)) < 1$ iff there exists a state x of \mathcal{M} and a set $Q \subseteq S$ s.t. $P_I(x) > 0$, $s \notin Q$ and the state (x, Q) is probable.

Proof. Let w be a word in $\overline{L(s)}$, then $s \notin \text{type}_A(w)$. By Lemma 20, $w \in L(Q)$ for some Q s.t. $s \notin Q$. This implies that $\overline{L(s)} \subseteq \cup_{s \notin Q} L(Q)$. By Theorem 2, for every Q s.t. $s \notin Q$ we have $L(Q) \subseteq \overline{L(s)}$, thus, $\overline{L(s)} = \cup_{s \notin Q} L(Q)$. Then, $P_{\mathcal{M}}(L(s)) < 1$ iff $P_{\mathcal{M}}(\overline{L(s)}) > 0$ iff there exists a set Q s.t. $s \notin Q$ and $P_{\mathcal{M}}(L(Q)) > 0$ iff there exists Q s.t. $s \notin Q$, a state x such that $P_I(x) > 0$, and $P(x, Q) > 0$. \square

B.1 Proof of Theorem 3

Theorem 3 Let C be an SCC of G . Then C is a BSCC of H iff it satisfies the following conditions:

1. C corresponds to a BSCC K of \mathcal{M} .
2. Every finite path of K is a projection of a path in C .
3. C contains a fulfilling path.

In the rest of the section we prove theorem 3. We start by showing that every *BSCC* of H satisfies the conditions of the theorem. Lemma 21 makes a connection between the paths of \mathcal{M} and the pathes of H .

Lemma 21. Let C be a BSCC of H . Let (x, Q) be a state in C , and let w be a path of \mathcal{M} that starts at x and has type Q . Then, with probability one, the augmented path of w is contained in C .

Proof. Let $\Theta = \{w | w \text{ starts at } x, \text{ has type } Q, \text{ and its augmented path is not contained in } C\}$. We prove that $P_{\mathcal{M}(x)}(\Theta) = 0$. First, note that all the augmented pathes of the paths in Θ start at (x, Q) . Since non-probable states do not have probable successors, for every word w in Θ there exists a unique i such that the i 'th state in the augmented path of w is in H and the $i + 1$ 'th state of the augmented path of w is not. We define

$$\Theta_i = \{w | w \text{ starts at } x, \text{ has type } Q, \text{ and its augmented path exits } C \text{ in its } i \text{'th state}\},$$

then we have $\Theta = \cup_{i=1}^{\infty} \Theta_i$. Thus, it is sufficient to prove that for every i we have $P_{\mathcal{M}(x)}(\Theta_i) = 0$. We further partition Θ_i according to the type Q' of the suffix of w that starts at w_{i+1} and the prefix $\rho = w_0, w_1, \dots, w_{i+1}$. Note that (w_{i+1}, Q') is not probable. For a set Q' of states and a finite path ρ of length $i + 1$, we define

$$\Theta_i^{Q', \rho} = \{w | Q' \text{ is the type of } w^{i+1}, \rho = w_0, w_1, \dots, w_{i+1}, \text{ and } P(w_{i+1}, Q') = 0\}$$

Then $\Theta_i \subseteq \cup_{\rho \in X^{i+1}, Q' \subseteq S} \Theta_i^{Q', \rho}$. For the set $\Theta_i^{Q', \rho}$ we have $P_{\mathcal{M}(x)}(\Theta_i^{Q', \rho}) \leq P_{\mathcal{M}(x)}(\rho) \cdot P(w_{i+1}, Q')$. Since $P(w_{i+1}, Q') = 0$, we have that $P_{\mathcal{M}(x)}(\Theta_i^{Q', \rho}) = 0$. This implies that $P_{\mathcal{M}(x)}(\Theta_i) = 0$, so $P_{\mathcal{M}(x)}(\Theta) = 0$. \square

Lemma 22. Let C be a BSCC of H that corresponds to an SCC K of \mathcal{M} . Then K is a BSCC of \mathcal{M} .

Proof. Let (x, Q) be a state in a BSCC C of H . Suppose K is not a BSSC of \mathcal{M} . Then a path ρ starting from $x \in K$ almost surely leaves K , meaning it has a suffix with no letters in K . Since (x, Q) is in H , with positive probability ρ has type Q . This implies that with positive probability the augmented path of ρ starts at (x, Q) and its projection has a suffix with no letters in K . This implies with positive probability the augmented path of ρ starts at (x, Q) and exits C , meaning exits H . This contradicts Lemma 21. \square

Lemma 23. *Let C be a BSCC of H that corresponds to a BSCC K of \mathcal{M} . Then every finite path in K is the projection of some path in C .*

Proof. Suppose that α is a finite path in K that is not the projection of a path in C . Let x be a state in K , and let (x, Q) be a state in C . Since (x, Q) is in C , with positive probability a path w of \mathcal{M} that starts at x has type Q . By ergodicity, a path that starts at x almost surely eventually takes α . Thus, with positive probability, a path w that starts at x has type Q and contains α . Let ρ be the augmented path of a path w that starts at x and has type Q . By the definition of augmented path, ρ starts at (x, Q) , and since the projection of ρ contains α , ρ exits C . This implies that with positive probability the augmented path of a path of \mathcal{M} that starts at x exits C , which contradicts Lemma 21. \square

We now prove that every BSCC of H contains a fulfilling path. We start by showing that every augmented path in G contains a fulfilling path.

Lemma 24. *Let w be a path in \mathcal{M} . Then the augmented path ρ_G of w contains a fulfilling path.*

Proof. Let Q_1 be the type of w . By Lemma 20, there exists an accepting run ρ_f of $\mathcal{A}^{(Q_1, Q_2, \emptyset, \emptyset)}$ on w . For every i , since ρ_f^i is a run of $\mathcal{A}^{(Q_1^i, Q_2^i, Q_3^i, Q_4^i)}$ on w^i , the first element Q_1^i of the i th state of ρ_f is $\text{type}_{\mathcal{A}}(w^i)$. Together with the definition of augmented path this implies that $\pi_1(\rho_f) = \pi_2(\rho_G)$. Since ρ_f is an accepting run, it contains infinitely many accepting states of the form $(Y_1, Y_2, \emptyset, \emptyset)$. Recall that from a state of the form $(Y_1, Y_2, \emptyset, \emptyset)$ all the outgoing transitions in \mathcal{A}_f enter states of the form $(Y'_1, Y'_2, (Y'_1 \setminus F), (Y'_2 \setminus \hat{F}))$. This implies that ρ_G contains a fulfilling path. \square

Lemma 25. *Let C be a BSCC of H , then C contains a fulfilling path.*

Proof. Let (x, Q) be a state in C . Since C is in H , (x, Q) is probable. Thus, with positive probability, a path that starts with x is of type Q . Lemma 21 implies that with probability one an augmented path of a path that starts in x and has a type Q is contained in C . Thus, with positive probability, the augmented path of a path that starts at x is contained in C . Since there is at least one outgoing path out of x , this implies that there exists a path that starts at x such that its augmented path is contained in C . Together with Lemma 24, this implies that C contains a fulfilling path. \square

This completes the proof that of the first direction of Theorem 3. In the rest of this section we prove the other direction of Theorem 3. Let C be an SCC of G , we prove that if C satisfies the conditions of the theorem, then C is a BSCC of H . Let C be an SCC of G that corresponds to a BSCC K of \mathcal{M} such that every finite path in K is a projection of a path in C . The next four lemmas prove that if C contains a fulfilling path, then C is a BSCC of H .

Lemma 26. *Let C be an SCC of G that corresponds to a BSCC K of \mathcal{M} such that every finite path of K is a projection of a path in C . Let ρ_C be an infinite path in C and let ρ_K be the projection of ρ_C . If ρ_K contains every finite path in K infinitely many times, then ρ_C contains every finite path in C infinitely many times.*

Proof. Every suffix of ρ_K contains every finite path in K infinitely many times. Assume that ρ_C contains some finite path α of C only finitely many times, then there exists a suffix ρ'_C of ρ_C such that its projection ρ'_K on \mathcal{M} contains every finite path in K infinitely often and ρ'_C does not contain α at all. Thus, it is enough to prove that ρ_C contains every finite path of C . Assume now to the contrary that there is a finite path α in C that is not contained in ρ_C . For every finite path ρ in C we define $\Xi(\rho)$ to be the set of finite paths ρ' in C that have the same projection on K as ρ and do not contain α . Let $\xi(\rho)$ be the set of states in C that are the last states of the paths in $\Xi(\rho)$.

We define a sequence $\alpha_0, \alpha_1, \dots$ of paths in C such that for every i such that $|\xi(\alpha_i)| \geq 1$, we have that $|\xi(\alpha_i)| > |\xi(\alpha_{i+1})|$, and α_i contains α . Since $\xi(\alpha_0)$ is a finite set, there exists n such that $|\xi(\alpha_n)| = 0$. Let β_i be the projection of α_i , for $i \geq 0$. Then all paths in C with projection β_n contain α . Since ρ_K contains β_n , we have that ρ_C contains α , contradiction.

We define $\alpha_0 = \alpha$. Let α_i be the i 'th path in the sequence. Let (x, Q) be a state in $\xi(\alpha_i)$, and let ρ be the path in $\Xi(\alpha_i)$ (so it has projection β_i) that ends in (x, Q) . We define α_{i+1} to be $\alpha\gamma\rho$ where γ is some path that connects α to ρ . Since C is an SCC, there exists such a path. Note that α_{i+1} has a suffix with projection β_i thus $\xi(\alpha_{i+1}) \subseteq \xi(\alpha_i)$, furthermore, the path with projection β_{i+1} that ends in (x, Q) contains α , thus $(x, Q) \notin \xi(\alpha_{i+1})$. \square

Lemma 19, Lemma 4, and the definition of G imply the following lemma.

Lemma 27. *Let $\rho_G = (x^0, Q^0), (x^1, Q^1), \dots$ be a finite or infinite path in G , then:*

1. *Let ρ_f be a run of \mathcal{A}_f such that $\pi_1(\rho_f) = \pi_2(\rho_g)$. Then ρ_f is a run of \mathcal{A}_f on $\pi_1(\rho_G)$.*
2. *The sequence $(Q^0, S \setminus Q^0, Q^0 \setminus F, (S \setminus Q^0) \setminus \hat{F}), (Q^1, S \setminus Q^1, Q^1 \setminus F, (S \setminus Q^1) \setminus \hat{F}), (Q^2, S \setminus Q^2, Q^2 \setminus F, (S \setminus Q^2) \setminus \hat{F}), \dots$ is a run of \mathcal{A}^f on x^0, x^1, \dots*

Lemma 28. *Let α be a fulfilling path in G . Let ρ_G be a path in G that starts at a state (x, Q_1) and contains α infinitely often. Then $\mathcal{A}_f^{(Q_1, Q_2, (Q_1 \setminus F), (Q_2 \setminus \hat{F}))}$ accepts the projection $\pi_1(\rho_G)$ of ρ_G on \mathcal{M} .*

Proof. We show an accepting run of $\mathcal{A}_f^{(Q_1, Q_2, (Q_1 \setminus F), (Q_2 \setminus \hat{F}))}$ on $\pi_1(\rho_G)$. Let $i_1 = 0$ and i_2, i_3, \dots be an infinite sequence of indices such that for every $j \geq 1$, we have that the subpath $\rho_G^{i_{2j}}, \rho_G^{i_{2j}+1}, \dots, \rho_G^{i_{2j+1}}$ of ρ_G is α . Let α_f be the path of \mathcal{A}_f that corresponds to α as defined in Definition 14.

We define a run of \mathcal{A}_f as follows, for every $j \geq 1$, the part of the run that starts at i_{2j} and ends at i_{2j+1} is α_f . For every $j \geq 0$ and $i_{2j+1} < k < i_{2j+2}$, the k 'th state of the run is $Q_1^k = \rho_G^k, Q_2^k = S \setminus Q_1^k, Q_3^k = Q_1^k \setminus F$, and $Q_4^k = Q_2^k \setminus \hat{F}$.

Lemma 27 implies that the path that we describe is a run of $\mathcal{A}_f^{(Q_1, Q_2, (Q_1 \setminus F), (Q_2 \setminus \hat{F}))}$ on the projection of ρ_G . Since ρ_G contains infinitely many fulfilling paths, the run contains infinitely many accepting states. \square

Lemma 29. Let C be an SCC of G that corresponds to a BSCC K of \mathcal{M} and satisfies the conditions below.

- Every finite path of K is a projection of a path in C .
- C contains a fulfilling path.

Then, C is a BSCC of H .

Proof. First we prove that C is contained in H . Since for every SCC C of G either $C \subseteq H$ or $C \cap H = \emptyset$, it is enough to prove that C contains at least one probable state. We prove that with probability one a path ρ_K that starts from a state x in K is a projection of a path ρ_C that starts at a state (x, Q_1) in C , such that $\mathcal{A}_f^{(Q_1, S \setminus Q_1, (Q_1 \setminus F), (S \setminus Q_1 \setminus \hat{F}))}$ accepts ρ_K , that is, $\text{type}_{\mathcal{A}}(\rho_K) = Q_1$. This implies that with probability one, a path that starts from a state x in K has type Q_1 such that (x, Q_1) is in C . This implies that at least one state in C is probable.

Let ρ_K be a path of K that starts at x . Let ρ_C be a path in C with projection ρ_K on K . Then, by ergodicity (see Section 2.2) ρ_K contains every finite path in K infinitely often. Lemma 26 implies that ρ_C contains every finite path in C infinitely often, and in particular it contains the fulfilling path infinitely often. Lemma 28 implies that $\mathcal{A}_f^{(Q_1, Q_2, (Q_1 \setminus F), (Q_2 \setminus \hat{F}))}$ accepts ρ_K .

Finally, we prove that C is a BSCC of H . Since C corresponds to a BSCC K of \mathcal{M} , every successor SCC C' of C in G corresponds to K too. Lemma 8 implies that there exists a finite path in K that is not the projection of a path in C' . By Lemma 23 C' is not a BSCC of H . Since C is a SCC of H and no successor SCC of C is a BSCC of H , we conclude that C is a BSCC of H \square

C Proofs for Section 5

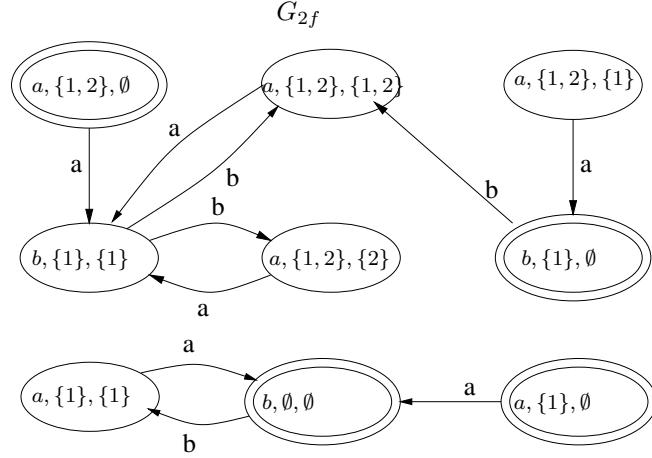
Lemma 30. Let ρ be a path in G_f , then the projection of the states of ρ on G is a path in G .

Proof. Let $((x, (Q_1, Q_2, Q_3, Q_4)), (x', (Q'_1, Q'_2, Q'_3, Q'_4)))$ be a transition in G_f , we prove that $((x, Q_1), (x', Q'_1))$ is a transition of G . The definition of G_f implies that $P_T(x, x') > 0$ and that $(Q'_1, Q'_2, Q'_3, Q'_4) \in \delta_f((Q_1, Q_2, Q_3, Q_4), x)$. Lemma 4 implies that (Q_1, x, Q'_1) is a transition in T_A . By the definition of G , we have that $((x, Q_1), (x', Q'_1))$ is a transition of G . \square

Lemma 9 An SCC C of G contains a fulfilling path iff there exists a path $(x_0, (Q_1^0, Q_2^0, Q_3^0, Q_4^0)), (x_1, (Q_1^1, Q_2^1, Q_3^1, Q_4^1)), \dots, (x_n, (Q_1^n, Q_2^n, \emptyset, \emptyset))$ in G_f such that the path $(x_0, Q_1^0), (x_1, Q_1^1), \dots, (x_n, Q_1^n)$ is contained in C , $Q_3^0 = Q_1^0 \setminus F$, and $Q_4^0 = Q_2^0 \setminus \hat{F}$.

Lemma 9 is followed directly from Lemma 30.

Example 6. We present in Figure 4 the product G_{2f} of \mathcal{A}_f and \mathcal{M}_2 which are presented in Figures 1 and 2. It is easy to see that the only SCC in G_2 that contains a fulfilling path is the SCC of $(a, \{1\})$ and (b, \emptyset) that contains the path $(a, \{1\}), (b, \emptyset)$ that correspond to the path $(a, \{1\}, \{1\}), (b, \emptyset, \emptyset)$ of G_{2f} . \square

**Fig. 4.** The graph G_{2f} .

D Exact Probability of $L(\mathcal{A})$

In order to compute the exact probability $P_M(L(\mathcal{A}))$, we prove that the graph H can be defined as a Markov chain that refines \mathcal{M} . First we define a Markov chain over H . Define the initial probability $P_I^H(x, Q)$ as $P_I(x) \cdot P(x, Q)$. Define the probability of a transition $(x, Q) \rightarrow (x', Q')$ to be

$$P_T^H((x, Q), (x', Q')) = \frac{P_T(x, x')P(x', Q')}{P(x, Q)}.$$

Lemma 31. *H is a Markov Chain.*

Proof. The sum of the initial probabilities in H is equal to

$$\sum_{(x, Q) \in G} P_I(x)P(x, Q) = \sum_x P_I(x) \sum_Q P(x, Q)$$

Since every path that starts at x has a unique type, we have $\sum_Q P(x, Q) = 1$. Thus,

$$\sum_x P_I(x) \sum_Q P(x, Q) = \sum_x P_I(x) = 1.$$

Next, we show that for every state in H , the probabilities of its outgoing transitions are sum to one. Consider $(x, Q) \in H$. Then, the sum of the probabilities of the outgoing transitions from (x, Q) is:

$$\sum_{(x, Q) \rightarrow (x', Q')} \frac{P_T(x, x')P(x', Q')}{P(x, Q)} = \frac{1}{P(x, Q)} \cdot \sum_{x \rightarrow x'} P_T(x, x') \cdot \sum_{Q \xrightarrow{x} Q'} P(x', Q')$$

Thus, it is enough to prove that

$$\sum_{x \rightarrow x'} P_T(x, x') \cdot \sum_{Q \xrightarrow{x} Q'} P(x', Q') = P(x, Q).$$

We can partition the probability that a path that starts at x is accepted by Q according to the second letter of the path. For every x' the probability that a path starts at x its second letter is x' and it is accepted by Q is $P_T(x, x') \cdot \sum_{Q \xrightarrow{x'} Q'} P(x', Q')$. \square

Next, we prove that H refines \mathcal{M} . We start by proving that for every regular language L we have $P_M(L) = P_H(L)$. We define a labelling function V for H such that $V(x, Q) = x$ thus both Markov chains are defined over the same alphabet.

Lemma 32. *For every ω -regular language L we have $P_M(L) = P_H(L)$.*

Proof. It is sufficient to show that \mathcal{M} and H agree on the basic cylindrical sets. That is, for every finite word $w = x_0, x_1, \dots, x_k$, we have $P_M(\Delta(w)) = P_H(\Delta(w))$. Recall that T_A is reverse deterministic. So for a fixed $w = x_0, \dots, x_k$ and Q_k , there exists a unique path $\rho = (x_0, Q_0), \dots, (x_k, Q_k)$ such that $V(\rho) = w$ and ρ ends in (x_k, Q_k) . We define $\rho^{w,Q}$ to be the unique path in H such that $V(\rho^{w,Q}) = w$ and $\rho_k^{w,Q} = Q$. So,

$$\begin{aligned} & P_H(\Delta(w)) \\ &= \sum_{V(\rho)=w} P_H(\rho), \text{ we partition according to the } k\text{'th state of } \rho \\ &= \sum_Q P_H(\rho^{w,Q}) \} \\ &= \sum_Q [P_I^H(\rho_0^{w,Q}) \prod_{0 \leq i < k} P_T^H(\rho_i^{w,Q}, \rho_{i+1}^{w,Q})] \\ &= \sum_Q [P_I(x_0) P(\rho_0^{w,Q}) \prod_{0 \leq i < q} P_T(x_i, x_{i+1}) \frac{P(\rho_{i+1}^{w,Q})}{P(\rho_i^{w,Q})}], \text{ definition of } P_T^H \\ &= \sum_Q [P_I(x_0) [\prod_{0 \leq i < k} P_T(x_i, x_{i+1})] P(\rho_k^{w,Q})] \\ &= \sum_Q [P_M(w) \cdot P(\rho_k^{w,Q})] \text{ since } \rho_k^{w,Q} = (x_k, Q), \\ &= P_M(w). \square \end{aligned}$$

In order to complete the proof that H refines \mathcal{M} we need to prove that $P_M(L(Q_1, Q_2, Q_3, Q_4)) = \sum_{x \in X} P_I^H(x, Q_1)$. This is proved in Lemma 33.

Lemma 33. $P_M(L((Q_1, Q_2, Q_3, Q_4))) = \sum_{x \in X} P_I^H(x, Q_1)$.

Proof. Lemma 32 implies that $P_M(\{w|w_0 = x \text{ and } w \text{ has type } Q_1\}) = P_H(\{w|w_0 = x \text{ and } w \text{ has type } Q_1\})$. We need to prove that $P_H(\{w|w_0 = x \text{ and } w \text{ has type } Q_1\}) = P_I^H(x, Q_1)$. It is enough to prove that a projection of a path that starts in (x, Q_1) is with

probability one of type Q_1 meaning is accepted with probability one by $\mathcal{A}_f^{(Q_1, Q_2, (Q_1 \setminus F), (Q_2 \setminus \hat{F}))}$. Let ρ be a path that starts at (x, Q_1) . By ergodicity, ρ reaches a BSCC of H and then pass through every finite path of the BSCC infinitely often. In particular, ρ pass infinitely often through a fulfilling path. Lemma 28 implies that $\mathcal{A}_f^{(Q_1, Q_2, (Q_1 \setminus F), (Q_2 \setminus \hat{F}))}$ accepts the projection of ρ on \mathcal{M} . \square

Given a state s of \mathcal{A} , we have $L(s) = \cup_{s \in Q} L(Q)$, which implies that $P_M(L(s)) = \sum_{s \in Q} \sum_{x \in X} P_I^H(x, Q)$. Thus, it left to calculate the exact probabilities of H . We need to compute for every state (x, Q) of H , the probability $P(x, Q)$. In [10] it is shown that these probabilities form a unique solution to the following linear system:

1. for every state (x, Q) of H we have $P(x, Q) = \sum_{((x, Q), (x', Q')) \in G} P_T(x, x') P(x', Q')$.
2. For every state x of \mathcal{M} we have $\sum_{Q \in 2^s} P(x, Q) = 1$.

Automatic Structures: Richness and Limitations

Bakhadyr Khoussainov

Department of Computer Science
University of Auckland, New Zealand
bmk@cs.auckland.ac.nz

Sasha Rubin

Department of Mathematics
University of Auckland, New Zealand
rubin@math.auckland.ac.nz

Andre Nies

Department of Computer Science
University of Auckland, New Zealand
andre@math.auckland.ac.nz

Frank Stephan

National ICT Australia
Sydney Research Laboratory at Kensington
fstephan@cse.unsw.edu.au

Abstract

This paper studies the existence of automatic presentations for various algebraic structures. The automatic Boolean algebras are characterised, and it is proven that the free Abelian group of infinite rank and many Fraïssé limits do not have automatic presentations. In particular, the countably infinite random graph and the universal partial order do not have automatic presentations. Furthermore, no infinite integral domain is automatic. The second topic of the paper is the isomorphism problem. We prove that the complexity of the isomorphism problem for the class of all automatic structures is Σ_1^1 -complete.

1 Introduction

Classes of infinite structures with nice algorithmic properties (such as decidable model checking) are of increasing interest in a variety of fields of computer science. One is in the theory of infinite state transition systems and the questions of their symbolic representations, model checking, specification and verification. Another is that of extending the framework of finite model theory to infinite models that have finite presentations. Also, string query languages in databases may be captured by (decidable) infinite string structures. Automatic structures are (usually) infinite relational structures whose domain and atomic relations can be recognised by finite automata operating synchronously on their input. Consequently, automatic structures have finite presentations and are closed under first order interpretability (as well as some of extensions of first order). Moreover, the model checking problem for

automatic structures is decidable. Hence automatic structures and tools developed for their study are well suited to these fields of computer science, see for instance [1].

From a computability and logical point of view, automatic structures are used to provide generic examples of structures with decidable theories, to investigate the relationship between automata and definability, and to refine the ideas and approaches in the theory of computable structures. This paper investigates the problem of characterising automatic structures in algebraic, model theoretic or logical terms.

This paper addresses two foundational problems in the theory of automatic structures. The first is that of providing *structure theorems* for classes of automatic structures. Fix a class \mathcal{C} of structures, closed under isomorphism. For instance, \mathcal{C} may be the class of groups or linear orders. A structure theorem should be able to distinguish whether a given member of \mathcal{C} has an automatic presentation or not; a special case of this is to know whether a given structure is automatically presentable or not. This usually concerns the interactions between the combinatorics of finite automata presenting structures and properties of the structures themselves. The second problem, which is related to the first, is the complexity of the *isomorphism problem* for classes of automatic structures. Namely fix a class of automatic structures \mathcal{C} . Given automatic presentations of two structures from \mathcal{C} , are the structures isomorphic?

With regard to the first problem, we provide new techniques for proving that some foundational structures in computer science and mathematics do not have automatic presentations. For example, we show that the

Fraïssé limits of many classes of finite structures such as finite partial orders or graphs do not have automatic presentations. This shows that the infinite random graph and universal partial order do not have automatic presentations. The idea is that the finite amount of memory intrinsic to finite automata presenting the structure can be used to extract algebraic and model theoretic properties (invariants) of the structure, and so used to classify such structures up to isomorphism. This line of research has indeed been successful in investigating automatic ordinals, linear orders, trees and Boolean algebras. For example we know a full structure theorem for the automatically presentable ordinals; namely, they are those strictly less than ω^ω [5]. We have partial structure theorems saying that automatic linear orders and automatic trees have finite Cantor-Bendixson rank [12]. In this paper we provide a structure theorem for the (infinite) automatic Boolean algebras; namely, they are those isomorphic to finite products of the Boolean algebra of finite and co-finite subsets of \mathbb{N} .

With regard to the second problem, it is not surprising that the isomorphism problem for the class of all automatic structures is undecidable [4]. The reason for the undecidability is that the configuration space of a Turing machine considered as a graph is an automatic structure, and the reachability problem in the configuration space is undecidable. Thus with some extra work as in [3] or [10] one can reduce the reachability problem to the isomorphism problem for automatic structures. In addition, the isomorphism problem for automatic ordinals [12] and Boolean algebras (Corollary 3.5) is decidable, for equivalence structures is Π_1^0 , and for configuration spaces of Turing machines is Π_3^0 -complete [13].

Hence it is somewhat unexpected that the complexity of the isomorphism problem for the class of automatic structures is Σ_1^1 -complete. The Σ_1^1 -completeness is proved by reducing the isomorphism problem for computable trees, known to be Σ_1^1 -complete [8], to the isomorphism problem for automatic structures.

The two problems are related in the following way. If one has a ‘nice’ structure theorem for a class \mathcal{C} of automatic structures, then one expects that the isomorphism problem for \mathcal{C} be computationally ‘reasonable’. For instance, as corollaries of the structure theorems for automatic ordinals and for automatic Boolean algebras, one obtains that their corresponding isomorphism problems are decidable. In contrast, the Σ_1^1 -completeness of the isomorphism problem of the class of all automatic structures tells us that the language of first order arithmetic is not powerful enough to give a

structure theorem for the class of all automatic structures. In other words we should not expect a ‘nice’ structure theorem for the class of all automatic structures.

Consequently the following approaches to studying automatic structures suggest themselves. Classify classes \mathcal{C} of automatic structures in terms of the complexity of the isomorphism problem for \mathcal{C} . Find structure theorems for automatic structures with restricted presentations that guarantee the decidability (or more generally Σ_n^0 -completeness) of the isomorphism problem. For instance, one may restrict the class of automatic graphs to those for which the reachability relation is regular, or decidable. These are not pursued in this paper.

Here is an outline of the rest of the paper. The next section is a brief introduction to the basic definitions. Section 3 provides some counting techniques sufficient to prove non-automaticity of many classical structures such as fields, integral domains and Boolean algebras. Section 4 provides a technique that is used to show non-automaticity of several structures such as the infinite random graph and the universal partial order. The last section is devoted to proving that the isomorphism problem for automatic structures is Σ_1^1 -complete. All unproved results are proved in the complete version of this paper.

Finally, we note that this paper does not deal with the tree automatic structures, a natural generalisation of structures presented by finite automata. Some of the techniques and results of this paper can be applied or extended to tree automatic structures in a natural way. In addition, techniques developed and results obtained in the study of structures presented by finite automata give rise to better understanding and deeper development of tree automatic structures.

2 Preliminaries

A thorough introduction to automatic structures can be found in [3] and [11]. We assume familiarity with the basics of finite automata theory though to fix notation the necessary definitions are included. A *finite automaton* \mathcal{A} over an alphabet Σ is a tuple (S, ι, Δ, F) , where S is a finite set of *states*, $\iota \in S$ is the *initial state*, $\Delta \subset S \times \Sigma \times S$ is the *transition table* and $F \subset S$ is the set of *final states*. A *computation* of \mathcal{A} on a word $\sigma_1\sigma_2\dots\sigma_n$ ($\sigma_i \in \Sigma$) is a sequence of states, say q_0, q_1, \dots, q_n , such that $q_0 = \iota$ and $(q_i, \sigma_{i+1}, q_{i+1}) \in \Delta$ for all $i \in \{0, 1, \dots, n-1\}$. If $q_n \in F$, then the computation is *successful* and we say that automaton \mathcal{A} *accepts* the word. The *language* accepted by the automaton \mathcal{A} is the set of all words accepted by \mathcal{A} . In

general, $D \subset \Sigma^*$ is *finite automaton recognisable*, or *regular*, if D is the language accepted by a finite automaton \mathcal{A} .

The following definitions extends recognisability to relations of arity n , called *synchronous n -tape automata*. A synchronous n -tape automaton can be thought of as a one-way Turing machine with n input tapes [7]. Each tape is regarded as semi-infinite having written on it a word in the alphabet Σ followed by an infinite succession of blanks, \diamond symbols. The automaton starts in the initial state, reads simultaneously the first symbol of each tape, changes state, reads simultaneously the second symbol of each tape, changes state, etc., until it reads a blank on each tape. The automaton then stops and accepts the n -tuple of words if it is in a final state. The set of all n -tuples accepted by the automaton is the relation recognised by the automaton. Here is a definition.

Definition 2.1 Write Σ_\diamond for $\Sigma \cup \{\diamond\}$ where \diamond is a symbol not in Σ . The convolution of a tuple $(w_1, \dots, w_n) \in \Sigma^{*n}$ is the string $\otimes(w_1, \dots, w_n)$ of length $\max_i |w_i|$ over alphabet $(\Sigma_\diamond)^n$ defined as follows. Its k 'th symbol is $(\sigma_1, \dots, \sigma_n)$ where σ_i is the k 'th symbol of w_i if $k \leq |w_i|$ and \diamond otherwise.

The convolution of a relation $R \subset \Sigma^{*n}$ is the relation $\otimes R \subset (\Sigma_\diamond)^{n*}$ formed as the set of convolutions of all the tuples in R . That is $\otimes R = \{\otimes w \mid w \in R\}$.

Definition 2.2 An n -tape automaton on Σ is a finite automaton over the alphabet $(\Sigma_\diamond)^n$. An n -ary relation $R \subset \Sigma^{*n}$ is finite automaton recognisable (in short FA recognisable) or regular if its convolution $\otimes R$ is recognisable by an n -tape automaton.

We now relate n -tape automata to structures. A *structure* \mathcal{A} consists of a countable set A called the *domain* and some relations and operations on A . We may assume that \mathcal{A} only contains relational predicates as the operations can be replaced with their graphs. We write $\mathcal{A} = (A, R_1^A, \dots, R_k^A, \dots)$ where R_i^A is an n_i -ary relation on A . The relation R_i are sometimes called basic or atomic relations. We assume that the function $i \rightarrow n_i$ is always a computable one.

Definition 2.3 A structure \mathcal{A} is automatic over Σ if its domain $A \subset \Sigma^*$ is finite automata recognisable, and there is an algorithm that for each i produces a finite automaton recognising the relation $R_i^A \subset (\Sigma^*)^{n_i}$. A structure is called automatic if it is automatic over some alphabet. If \mathcal{B} is isomorphic to an automatic structure \mathcal{A} , then call \mathcal{A} an automatic presentation of \mathcal{B} and say that \mathcal{B} is called automatically presentable (over Σ).

An example of an automatic structure is the word structure $(\{0, 1\}^*, L, R, E, \preceq)$, where for all $x, y \in \{0, 1\}^*$, $L(x) = x0$, $R(x) = x1$, $E(x, y)$ iff $|x| = |y|$, and \preceq is the lexicographical order. The configuration graph of any Turing machine is another example of an automatic structure. Examples of automatically presentable structures are $(\mathbb{N}, +)$, (\mathbb{N}, \leq) , (\mathbb{N}, S) , the group $(\mathbb{Z}, +)$, the order on the rationals (\mathbb{Q}, \leq) , and the Boolean algebra of finite or co-finite subsets of \mathbb{N} . Note that every finite structure is automatically presentable. We use the following important theorem without reference.

Theorem 2.4 [11] Let \mathcal{A} be an automatic structure. There exists an algorithm that from a first order definition ϕ in \mathcal{A} of a relation R produces an automaton recognising R .

3 Proving Non-Automaticity via Counting

The first technique for proving non-automaticity was presented in [11] and later generalised in [3]. The technique is based on a pumping argument and exhibits the interplay between finitely generated (sub) algebras and finite automata. We briefly recall the technique for completeness.

A relation $R \subset A^{k+l}$ is called *locally finite* if for every \bar{a} (of size k) there are at most a finite number of \bar{b} (of size l) such that $(\bar{a}, \bar{b}) \in R$. For $\bar{b} = (b_1, \dots, b_m)$, write $b \in \bar{b}$ to mean $b = b_i$ for some i .

We start with the following elementary but important lemma that will later be used without reference to it.

Lemma 3.1 Suppose that a relation $R \subset A^{k+l}$ is locally finite and FA recognisable. There exists a constant p such that for every $(\bar{x}, \bar{y}) \in R$ it is the case that

$$\max\{|y| \mid y \in \bar{y}\} - \max\{|x| \mid x \in \bar{x}\} \leq p.$$

Assume \mathcal{A} is an automatic structure in which each atomic relation R_i is a graph of a function f_i , $i = 1, \dots, n$. Let a_1, a_2, \dots be a sequence of some elements of A such that the relation $\{(a_i, a_j) \mid i \leq j\}$ is regular. Consider the sequence $G_1 = \{a_1\}$, $G_{n+1} = G_n \cup \{a_{n+1}\} \cup \{f_i(\bar{a}) \mid \bar{a} \in G_n, i = 1, \dots, n\}$. By the lemma above there is a constant C such that the length of all elements in G_n is bounded by $C \cdot n$. Therefore the number of elements in G_n is bounded by $2^{O(n)}$. Some combinatorial reasoning combined with this observation can now be applied to provide examples of structures with no automatic presentations, see [3] and

[11]. For example, these include: (a) The free group on $k > 1$ generators; (b) The structure $(\mathbb{N}, |)$; (c) The structure (\mathbb{N}, p) , where $p : \mathbb{N}^2 \rightarrow \mathbb{N}$ is a pairing function; (d) The term algebra generated by finitely many constants with at least one non-unary atomic operation¹. Note all these structures have decidable first order theory.

In the next sections we provide other more intricate techniques for showing that particular structures do not have automatic presentations. We then apply those techniques to give a characterisation of Boolean algebras that have automatic presentations. We also prove that (\mathbb{Q}^+, \times) has no automatic presentation and show that no infinite integral domain (in particular no infinite field) has an automatic presentation. We also study automaticity of some Fraïssé limits.

3.1 Automatic Boolean algebras

All finite Boolean algebras are automatic. Thus, in this section we deal with infinite countable Boolean algebras only. Our goal in this section is to give a full characterisation of infinite automatic Boolean algebras. Our characterisation can then be applied to show that the isomorphism problem for automatic Boolean algebras is decidable. Compare this with the result from computable algebra that the isomorphism problem for computable Boolean algebras is Σ_1^1 -complete [8].

Recall that a Boolean algebra $\mathcal{B} = (B, \cup, \cap, \setminus, \mathbf{0}, \mathbf{1})$ is a structure, where \cap and \cup and \setminus operations satisfy all the basic properties of the set-theoretic intersection, union, and complementation operations; In \mathcal{B} the relation $a \subseteq b \iff a \cap b = a$ is a partial order in which $\mathbf{0}$ is the smallest element, and $\mathbf{1}$ is the greatest element. The complement of an element $b \in B$ is $\mathbf{1} \setminus b$ and is denoted by \bar{b} .

The following lemma is useful.

Lemma 3.2 *Suppose \mathcal{B} is an automatic boolean algebra and let $n \in \mathbb{N}$. There exists a constant $e \in \mathbb{N}$ such that for every finite set S of n elements of B , the length of the element US is at most*

$$\max\{|s|\} + e \log n$$

where the maximum varies over $s \in S$.

¹Thus, elements of the term algebra are all the ground terms, and the operations are defined in a natural way: the value of a function f of arity n from the language on ground terms g_1, \dots, g_n is $f(g_1, \dots, g_n)$.

Proof We calculate a bound on $|US|$ by considering a ‘computation tree’ computing US . Let h be the least integer greater than or equal to $\log n$. Denote by B_h the full binary tree of height h . It has at least n leaves. Label the nodes of B_h as follows. The leaves are labelled by elements of S in such a way that every element of S is the label of at least one leaf. Suppose we have labelled all the elements on i ’th level of the tree, x_1, \dots, x_k , identifying the nodes of the trees with their labels. Label the $i - 1$ ’st level as follows. For every odd $j < k$, label the parent of x_j and x_{j+1} by $x_j \cup x_{j+1}$. This completes the labelling of the tree B_h . Note that the label of the root of the tree is US . By Lemma 3.1 there is a constant d such that for every $a, b \in B$ it holds that $|a \cup b| \leq \max|a|, |b| + d$. Then by induction if $x \in B$ is a label of a node on the i ’th level of the tree B_h then $|x| \leq \max|s| + d(h - i)$. Setting $i = 0$ and noting that $h \leq 1 + \log n$ it holds that $|US| \leq \max|s| + (d + 1) \log n$, as required. \square

A linearly ordered set determines a Boolean algebra in a natural way described as follows. Let $\mathcal{L} = (L, \leq)$ be a linearly ordered set. An interval is a subset of L of the form $[a, b] = \{x \mid a \leq x < b\}$, where $a, b \in L \cup \{\infty\}$. The **interval algebra** denoted by $\mathcal{B}_{\mathcal{L}}$ is the collection of all finite unions of intervals of \mathcal{L} , with the usual set-theoretic operations of intersection, union and complementation. Every interval algebra is a Boolean algebra. Moreover for every countable Boolean algebra \mathcal{A} there exists an interval algebra $\mathcal{B}_{\mathcal{L}}$ isomorphic to \mathcal{A} . We write $\mathcal{L}_1 \times \mathcal{L}_2$ for the ordered sum $\sum_{i \in \mathcal{L}_1} \mathcal{L}_2$. The proof of the following lemma is straightforward.

Lemma 3.3 *The interval Boolean algebras $\mathcal{B}_{i \times \omega}$, where i is positive integer, all have automatic presentations.*

An **atom** in a Boolean algebra is a non-zero element a such that for every $b \leq a$ we have $a = b$ or $b = \mathbf{0}$. Assume that \mathcal{B} is an automatic Boolean algebra not isomorphic to any of the algebras $\mathcal{B}_{i \times \omega}$. Call two elements $a, b \in B$ **F-equivalent** if the element $(a \cap \bar{b}) \cup (b \cap \bar{a})$ is a union of finitely many atoms. Factorise \mathcal{B} with respect to the equivalence relation. Denote the factor algebra by \mathcal{B}/F . Due to the assumption on \mathcal{B} the algebra \mathcal{B}/F is not finite. Call x in \mathcal{B} **large** if its image in \mathcal{B}/F is not a finite union of atoms or $\mathbf{0}$. For example the element $\mathbf{1}$ is large in \mathcal{B} because \mathcal{B} is not isomorphic to $\mathcal{B}_{i \times \omega}$. Call an element x in \mathcal{B} **infinite** if there are infinitely many elements below it. Say that x **splits** y , for $x, y \in B$, if $x \cap y \neq \mathbf{0}$ and $\bar{x} \cap y \neq \mathbf{0}$. For every large element $l \in B$ there exists an element $x \in B$ that splits l such that $x \cap l$ is large and $\bar{x} \cap l$ is infinite. Also for every infinite element $i \in B$ there

exists an element $x \in B$ that splits i such that either $x \cap i$ or $\bar{x} \cap i$ is infinite.

Now we construct a sequence T_n of trees and elements $a_\sigma \in B$ corresponding to elements $\sigma \in T_n$ as follows. The tree T_n will be a set of binary strings closed under prefixes. Therefore it suffices to define leaves of T_n . Initially, we set $T_0 = \{\lambda\}$ and $a_\lambda = \mathbf{1}$. Assume that T_n has been constructed. By induction hypothesis we may assume that the leaves of T_n satisfy the following properties: (1) There exists at least one leaf σ such that a_σ is large in B . Call the element a_σ leading in T_n . (2) There exist n leaves $\sigma_1, \dots, \sigma_n$ such that each a_{σ_i} is an infinite element in B . Call these elements sub-leading elements. (3) The number of leaves in T_n is greater than or equal to $n \cdot (n+1)/2$. (4) For every pair of leaves x, y of T_n it holds that $x \cap y = \mathbf{0}$.

For each leaf $\sigma \in T_n$ proceed as follows:

1. If σ is leading then find the first length lexicographical b that splits a_σ such that both $a_\sigma \cap b$ and $a_\sigma \cap \bar{b}$ are infinite and one of them is large. Put $\sigma 0$ and $\sigma 1$ into T_{n+1} . Set $a_{\sigma 0} = a_\sigma \cap b$ and $a_{\sigma 1} = a_\sigma \cap \bar{b}$.
2. If σ is a sub-leading then find the first length lexicographical b that splits a_σ such that one of $a_\sigma \cap b$ or $a_\sigma \cap \bar{b}$ is infinite. Put $\sigma 0$ and $\sigma 1$ into T_{n+1} . Set $a_{\sigma 0} = a_\sigma \cap b$ and $a_{\sigma 1} = a_\sigma \cap \bar{b}$.

Thus, we have constructed the tree T_{n+1} and elements a_σ corresponding to the leaves of the tree. Note that the inductive hypothesis holds for T_{n+1} . This completes the definition of the trees.

There exists a constant C_1 such that $|a_{\sigma \epsilon}| \leq |a_\sigma| + C_1$ for all defined elements a_σ . Now for every n consider the set $X_n = \{a_\sigma \mid \sigma \text{ is a leaf of } T_n\}$. There exists a constant C_2 such that for all $x \in X_n$ we have $|x| \leq C_2 \cdot n$. Therefore $X_n \subset \Sigma^{C_2 \cdot n}$ and the number of leaves in T_n is greater than or equal to $n \cdot (n+1)/2$. Now for every pair of elements a, b in X_n we have $a \cap b = \mathbf{0}$. Therefore the number of elements of the Boolean algebra generated by the elements in X_n is $2^{|X_n|}$. Now let $Y = \{b_1, \dots, b_k\} \subset X_n$. Consider the element $\cup Y = b_1 \cup \dots \cup b_k$. By Lemma 3.2 there exists a constant C_3 such that $|\cup Y| \leq C_3 \cdot n$. This gives us a contradiction because the number of elements generated by elements of X_n clearly exceeds the cardinality of $\Sigma^{O(n)}$.

Thus, we have proved the following theorem characterising infinite automatic Boolean algebras.

Theorem 3.4 *A Boolean algebra has an automatic presentation if and only if it is isomorphic to \mathcal{B}_α for some ordinal $\alpha < \omega^2$.*

The proof of the following is straightforward.

Corollary 3.5 *It is decidable whether two automatic Boolean algebras are isomorphic.*

3.2 Commutative monoids and Abelian groups

We prove that (\mathbb{Q}^+, \times) , or equivalently, the free Abelian group of rank ω , is not automatic. The following generalises Lemma 3.2 and is proved the same way.

Lemma 3.6 *Suppose (M, \times) is an automatic monoid. There exists a constant $k \in \mathbb{N}$ such that for every $n > 0$ and every sequence $s_1, \dots, s_n \in M$, the length of the element $\prod_i s_i$ is at most*

$$\max_i \{|s_i|\} + k \log n.$$

Theorem 3.7 *Let (M, \times) be a monoid containing (\mathbb{N}, \times) as a submonoid. Then (M, \times) is not automatic.*

Proof Assume for a contradiction that an automatic presentation of M is given. Let a_0, a_1, \dots be the prime numbers, viewed as elements of M , and listed in length-lexicographical order (with respect to this presentation of M). Let r_n be such that a_0, \dots, a_{r_n-1} are the primes of length at most n . Let

$$F_n = \{\prod_{i: 0 \leq i < r_n} a_i^{\beta_i} : 0 \leq \beta_i < 2^n\}.$$

By Lemma 3.6, each term $a_i^{\beta_i}$ has length at most $n + k \log \beta_i \leq n(1+k)$. Again by the lemma, each product has length at most $n(1+k) + k \log r_n$. Since all the products are distinct,

$$2^{nr_n} \leq |F_n| \leq |\Sigma|^{(1+k)n + k \log r_n}.$$

Thus $nr_n \leq \log |\Sigma|[(1+k)n + k \log r_n]$ and $r_n \in O(\log r_n)/n$, a contradiction because r_n goes to infinity. \square

Corollary 3.8 *(\mathbb{Q}^+, \times) is not automatically presentable.*

Corollary 3.9 *The monoid $(M_k(\mathbb{N}), \cdot)$, where $M_k(\mathbb{N})$ is the set of $k \times k$ matrices with entries from \mathbb{N} , is not automatically presentable.*

For a prime p , let \mathbb{Q}_p be the additive group of rationals of the form z/p^m , z an integer, $m \in \mathbb{N}$, and let C_p be the Prüfer group \mathbb{Q}_p/\mathbb{Z} . Using representations to base p , it is easy to give automatic presentations of these groups. Hence finite direct sums of those groups are also automatic. The proof of the following uses similar methods.

Theorem 3.10 *Let $A^{(\omega)}$ denote the direct sum of infinitely many copies of the group A . The infinite direct sums $\mathbb{Q}_p^{(\omega)}$ and $C_p^{(\omega)}$ are not automatically presentable.*

3.3 Integral domains

In our next result we prove that no infinite integral domain is automatically presentable. The following definition and lemma will be used in the next section as well.

Definition 3.11 Suppose \mathcal{D} is a structure over alphabet Σ . Write D_n for $D \cap \Sigma^{\leq n}$; that is the elements of D of length at most n . Write S_n for $\{x \in \Sigma^n \mid \exists z \in \Sigma^* \wedge xz \in D\}$; that is prefixes of length n of words in the domain.

Lemma 3.12 If $D \subset \Sigma^*$ is a regular language then

1. $|S_n| \in O(|D_n|)$ and
2. $|D_{n+k}| \in \Theta(|D_n|)$ for every constant $k \in \mathbb{N}$.

Proof Suppose the automaton recognising D has c states. Then for $x \in S_n$ there exists $z \in \Sigma^*$ with $|z| \leq c$ such that $xz \in D$ (\dagger). If $n \geq c$ then $|S_n| \leq |\Sigma|^c \times |S_{n-c}|$ since the map associating $x \in S_n$ with the word consisting of the first $n - c$ letters of x , is $|\Sigma|^c$ -to-one. But by using (\dagger) we see that $|S_{n-c}| \leq |D_n|$. So $|S_n| \leq |\Sigma|^c \times |D_n|$. This completes the first part.

Fix $k \in \mathbb{N}$. The mapping associating $x \in D_{n+k}$ to the prefix of x of length n is $|\Sigma|^k$ -to-one. Hence $|D_{n+k}| \leq |\Sigma|^k \times |S_n| \leq |\Sigma|^k \times |\Sigma|^c \times |D_n|$. Since $D_n \subset D_{n+k}$ one has that $|D_n| \leq |D_{n+k}|$ and $|D_{n+k}| \in O(|D_n|)$. This completes the second part. \square

Recall that an integral domain $(D, +, \cdot, 0, 1)$ is a commutative ring with identity such that $x \cdot y = 0$ only if $x = 0$ or $y = 0$. For example every field is an integral domain.

Theorem 3.13 No infinite integral domain is automatically presentable.

Proof Suppose that $(D, +, \cdot, 0, 1)$ is an infinite automatic integral domain. For each $n \in \mathbb{N}$ recall that $D_n = \{u \in D \mid |u| \leq n\}$. We claim that there exists an x in D such that for all $a, b, a', b' \in D_n$ the condition $a \cdot x + b = a' \cdot x + b'$ implies that $a = a'$ and $b = b'$. We say such x separates D_n . Indeed, assume that such an x does not exist. Then for each $x \in D$ there exist $a, b, a', b' \in D_n$ such that $a \cdot x + b = a' \cdot x + b'$ but $(a, b) \neq (a', b')$. Hence, since D_n is finite, there exist $a, b, a', b' \in D_n$ such that $a \cdot x + b = a' \cdot x + b'$ but $(a, b) \neq (a', b')$ for infinitely many x . Thus, for infinitely many x we have $(a - a') \cdot x = b' - b$. But $a \neq a'$ for otherwise also $b = b'$, contrary to assumption. Also there exist distinct x_1 and x_2 such that $(a - a') \cdot x_1 = (a - a') \cdot x_2$. Since D is an integral

domain we conclude that $x_1 = x_2$ which is a contradiction.

For each D_n we can select the length-lexicographically first x_n separating D_n . Now the set $E_n = \{y \mid \exists a, b \in D_n [y = ax_n + b]\}$ has at least $|D_n|^2$ many elements. However by Lemma 3.1 there exists a constant C such that $E_n \subset D_{n+C}$, and by Lemma 3.12 the number of elements in D_{n+C} is in $O(|D_n|)$. Thus, we have a contradiction. The theorem is proved. \square

Corollary 3.14 No infinite field is automatically presentable.

4 Non-automaticity of some Fraïssé Limits

In this section we provide some methods for proving non-automaticity of structures. These methods are then applied to prove that some Fraïssé limits do not have automatic presentation. We already know that the atomless Boolean algebra does not have an automatic copy. Below are examples of Fraïssé limits that have automatic presentations.

Example 4.1 The linear order of rational numbers has an automatic presentation. In fact it is straightforward to check that $(\{0, 1\}^* \cdot 1, \preceq_{lex})$ is an automatic presentation of (Q, \leq) .

Example 4.2 Let $U = \{u \mid u \in \{0, 1\}^* \cdot 1, |u| \text{ is even}\}$. The structure $(\{0, 1\}^* \cdot 1; \preceq_{lex}, U)$ is the Fraïssé limit for the class K of all finite linear orders with one unary predicate.

Let \mathcal{A} be an automatic structure over the alphabet Σ . The **standard approximation** of this structure is the sequence $A_0 \subseteq A_1 \subseteq A_2 \subseteq \dots$ such that $A_n = \{v \in A \mid |v| \leq n\}$. Let $\Phi(x, y)$ be a two variable formula in the language of this structure. We do not exclude that $\Phi(x, y)$ has a finite number of parameters from the domain of the structure. Now for each $y \in A$ and $n \in \mathbb{N}$ we define the following function $c_{n,y} : A_n \rightarrow \{0, 1\}$:

$$c_{n,y}(x) = \begin{cases} 1 & \text{if } \mathcal{A} \models \Phi(x, y); \\ 0 & \text{if } \mathcal{A} \models \neg \Phi(x, y). \end{cases}$$

In the next theorem we count the number of functions $c_{n,y}$ (which clearly depends on the given formula Φ) using the fact that \mathcal{A} is an automatic structure.

Theorem 4.3 If \mathcal{A} is automatic, then the number of functions $c_{n,y}$ is in $O(|A_n|)$.

Proof Let (Q, ι, ρ, F) be a deterministic automaton recognising the relation $\otimes\Phi = \{\otimes(x, y) \mid \mathcal{A} \models \Phi(x, y)\}$. Fix $n \in \mathbb{N}$. We need to find a constant k such that for all $y \in A$, the number of functions $c_{n,y}$ is at most $k|A_n|$. For $y \in A$ define the function $T_y : A_n \rightarrow Q$ and the subset Q_y of Q as follows. Let $\otimes(x, y)$ be $\sigma_1 \dots \sigma_k$. Then:

$$T_y(x) = \begin{cases} \rho(\iota, \otimes(x, y)) & \text{if } k \leq n; \\ \rho(\iota, \sigma_1 \dots \sigma_n) & \text{if } k > n. \end{cases}$$

Thus, $T_y(x)$ is the state resulted by reading the whole input (x, y) if $|y| \leq n$, and otherwise $T_y(x)$ is the state obtained by reading the first n symbols of the input $\otimes(x, y)$. The set Q_y is defined in a similar manner as follows. If $k \leq n$ then $Q_y = \{\rho(\iota, \otimes(x, y)) \mid x \in A_n \text{ \& } \rho(\iota, \otimes(x, y)) \in F\}$; If $k > n$ then $Q_y = \{s \mid \rho(s, \sigma_{n+1} \dots \sigma_k) \in F\}$.

Note that $\sigma_{n+1} \dots \sigma_k = \otimes(\epsilon, z)$ for some $z \in \Sigma^*$ in case $k > n$. We claim that if $(T_{y_1}, Q_{y_1}) = (T_{y_2}, Q_{y_2})$ then $c_{n,y_1} = c_{n,y_2}$. Assume that $c_{n,y_1}(x) \neq c_{n,y_2}(x)$ for some $x \in A_n$. So without loss of generality we may assume that $\mathcal{A} \models \Phi(x, y_1)$ and $\mathcal{A} \models \neg\Phi(x, y_2)$. If $T_{y_1}(x) \neq T_{y_2}(x)$ then we are done. Otherwise there exists a state q which is in Q_{y_1} but not in Q_{y_2} . This contradicts the assumption that $(T_{y_1}, Q_{y_1}) = (T_{y_2}, Q_{y_2})$. Thus, the number of functions of type $c_{n,y}$ is bounded by the number of pairs of type (T_y, Q_y) .

Recall the set $S_n = \{v \in \Sigma^n \mid \exists z(z \in \Sigma^* \wedge xz \in A)\}$. If $y \in A$ and $|y| > n$ then there exist v and z such that v is a prefix of y , $v \in S_n$ and $y = vz$. Moreover, $T_y = T_v$. Therefore, the number of pairs of type (T_y, Q_y) is bounded by $(|A_n| + |S_n|) \times 2^{|Q|}$. However by Lemma 3.12, $|S_n| \in O(|A_n|)$. So, finally we get that for $n \geq m$, the number of functions of type $c_{n,y}$ is bounded by the number of pairs of type (T_y, Q_y) which is bounded by

$$(|A_n| + |S_n|) \times 2^{|Q|} \in O(|A_n|) \times 2^{|Q|}.$$

We have proved the theorem. \square

Now we give several applications of this theorem. First we apply the theorem to random graphs. An explicit description of a random graph is the following. The set V of vertices is \mathbb{N} . The set of edges is defined as follows. Put an edge between n and m if in the binary representation of n , the term 2^m has coefficient 1.

Corollary 4.4 independently [6] *The random graph has no automatic presentation.*

Proof The random graph has the following algebraic property: For every disjoint partition X_1, X_2 of every finite set Y of vertices there exists a vertex y such that

$(y, x_1) \in E$ and $(y, x_2) \notin E$ for all $x_1 \in X_1$ and $x_2 \in X_2$.

Let (A, E) be an automatic presentation of the random graph. Consider the standard approximation A_0, A_1, A_2, \dots of it. For every partition X_1, X_2 of set A_n of vertices there exists a vertex y such that $(x_1, y) \in E$ and $(x_2, y) \notin E$ for all $x_1 \in X_1$ and $x_2 \in X_2$. Hence, for every n , we have $2^{|A_n|}$ number of functions of type $c_{n,y}$, contradicting Theorem 4.3. Hence the random graph has no automatic presentation. \square

Corollary 4.5 *Let \mathcal{A} be a random structure of a signature L , where L contains at least one non-unary symbol. Then \mathcal{A} does not have an automatic presentation.*

Proof Let R be a non-unary predicate of L of arity k . Consider the following formula $E(x, y) = R(x, y, a_1, \dots, a_{k-2})$, where a_1, \dots, a_{k-2} are fixed constants from the domain of \mathcal{A} . It is not hard to prove that (A, E) is isomorphic to the random graph. But if \mathcal{A} is automatically presentable then so is the random graph (A, E) , hence contradicting the previous corollary. \square

The next goal is to show that the K_p -free random graph has no automatic presentation. For this one needs to have a finer analysis than the one for the random graph. Let $\mathcal{G} = (V, E)$ be a finite graph. Denote by $\Delta(\mathcal{G})$ the maximum degree over all the vertices of \mathcal{G} . Call a subgraph \mathcal{G} with no edges an *independent* graph. Let $\alpha(\mathcal{G})$ be the number of vertices of a largest independent subgraph of \mathcal{G} . We need the following combinatorial lemma.

Lemma 4.6 *Let $\mathcal{G} = (V, E)$ be a finite graph. Then each of the following is true:*

1. $\alpha(\mathcal{G}) \geq |V| / (\Delta(\mathcal{G}) + 1)$.
2. For all $p \geq 3$, if \mathcal{G} is K_p -free then $\alpha(\mathcal{G}) \geq \sqrt[p-1]{|V|} - 1$.

Corollary 4.7 *For $p \geq 3$, the K_p -free random graph is not automatically presentable.*

Proof Fix $p \geq 3$ and let (A, E) be an automatic copy of the K_p -free random graph over alphabet Σ . Recall that $A_n = A \cap \Sigma^{\leq n}$. Let \mathcal{G} be an independent subgraph of A_n of size at least $\sqrt[p-1]{|A_n|} - 1$ as in the lemma. It has $2^{|\mathcal{G}|}$ subsets. By the property of the K_p -free random graph, for every K_{p-1} -free subset $K \subset A_n$ there exists an $x \in A$ that is connected to every vertex in K and no vertex in $A_n \setminus K$. So for a fixed $n \in \mathbb{N}$, the number of functions $c_{n,y}$ as y varies over A , is at least $2^{\sqrt[p-1]{|A_n|} - 1}$, contradicting Theorem 4.3. \square

As the fourth application we prove that the random partial order \mathcal{U} does not have an automatic presentation. For the proof we need the following combinatorial result that connects the size of a finite partial order (B, \leq) with the cardinalities of its chains and anti-chains.

Lemma 4.8 (Dilworth) *Let (B, \leq) be a finite partial order of cardinality n . Let a be the size of largest anti-chain in (B, \leq) and let c be the size of the largest chain in (B, \leq) . Then $n \leq ac$.*

Corollary 4.9 *The random partial order $\mathcal{U} = (U, \leq)$ has no automatic presentation.*

Proof The universal partial order has the following property (see [9]):

1. If Z is a *finite* anti-chain of \mathcal{U} and X and Y partition Z then there exists an element $z \in U$ such that for every $x \in X$, $z > x$ and for every $y \in Y$, element z is not comparable with y .
2. If Z is a *finite* chain of \mathcal{U} with least element x and largest element y then there exists an element $z \in U$ such that $z > x$ and $z < y$ and z is not comparable with every $v \in X \setminus \{x, y\}$.

Assume that \mathcal{U} has an automatic presentation (A, \leq) . Consider the standard approximation A_0, A_1, A_2, \dots of (A, \leq) . The formula $\Phi(x, y)$ is $x \leq y \vee y \leq x$. Now let us take A_n .

Let Z be an anti-chain of A_n . Consider a subset X of Z . There exists an element $y \in U$ such that for every $x \in X$, $y > x$ and for every $x' \notin A_n \setminus X$, element y is not comparable with x' . From this we conclude that

$$(*) \quad \#(\text{functions of type } c_{n,y}) \geq 2^{|Z|}.$$

Let Z is a *finite* chain of \mathcal{P} with least element v and largest element w then there exists an element $y \in U$ such that $y > v$ and $v < w$ and y is not comparable with x for every $x \in X$. From this we conclude that

$$(**) \quad \#(\text{functions of type } c_{n,y}) \geq |Z|^2.$$

Let X be the largest anti-chain and Y be the largest chain of A_n with cardinalities a and c , respectively. By Dilworth Lemma above, $(*)$ and $(**)$ we derive that $|A_n|^2 < c^2 \cdot a^2$ which in turn is less than

$$\begin{aligned} \#(\text{functions of type } c_{n,y}) &\times \\ \log^2[\#(\text{functions of type } c_{n,y})]. \end{aligned}$$

This bound clearly contradicts the statement of Theorem 4.3. The corollary is proved. \square

The following is another application of Theorem 4.3. It was observed by Leonid Libkin, though originally proven using growth level of string lengths in [3].

Corollary 4.10 *The pairing algebra (A, p) has no automatic presentation.*

5 The Isomorphism Problem

The results in the previous sections give us hope that one can characterise automatic structures for certain classes of structures, e.g. Boolean algebras. However, in this section we prove that the isomorphism problem is Σ_1^1 -complete, thus showing that the problem is as hard as possible when considering the class of *all* automatic structures. Then **complexity of the isomorphism problem** for automatic structures consists in establishing the complexity of the set $\{(\mathcal{A}, \mathcal{B}) \mid \mathcal{A}$ and \mathcal{B} are automatic structures and \mathcal{A} is isomorphic to $\mathcal{B}\}$.

Let \mathcal{M} be a Turing machine over input alphabet Σ . Its configuration graph $C(\mathcal{M})$ is the set of all configurations of \mathcal{M} , with an edge from c to d if T can move from c to d in a single transition. The following is an easy lemma:

Lemma 5.1 *For every Turing machine T the configuration graph $C(T)$ is automatic. Further, the set of all vertices with outdegree (indegree) 0 is FA-recognisable.*

Definition 5.2 *A Turing machine \mathcal{R} is reversible if every vertex in $C(\mathcal{R})$ has both indegree and outdegree at most one.*

Now let \mathcal{R} be a reversible Turing machine. Consider its configuration space $C(\mathcal{R})$. The machine \mathcal{R} can be modified so that it only halts in an accepting state; so, instead of halting in a rejecting state, it loops forever. Let x be a configuration of \mathcal{R} . Consider the sequence: $x = x_0, x_1, x_2, \dots$ such that $(x_i, x_{i+1}) \in E$, where E is the edge relation of the configuration space. Call this sequence the **chain defined by x** . We say that x is the **base** of chain X . If x does not have a predecessor then the chain defined by x is maximal. Since \mathcal{R} is reversible, the configuration space $C(\mathcal{R})$ is a disjoint union of maximal chains such that each chain is either finite, or isomorphic to (\mathbb{N}, S) , or isomorphic to (\mathbb{Z}, S) , where $(x, y) \in S$ iff $y = x + 1$. It is known that every Turing machine can be converted into an equivalent reversible Turing machine (see for example [2]). Our next lemma states this fact and provides some additional structural information about the configuration space of reversible Turing machines:

Lemma 5.3 *A deterministic Turing machine can be converted into an equivalent reversible Turing machine \mathcal{R} such that every maximal chain in $C(\mathcal{R})$ is either finite or isomorphic to (\mathbb{N}, S) .*

Denote by \mathbb{N}^* the set of all finite strings from \mathbb{N} . A set $T \subset \mathbb{N}^*$ is a *special tree* if T is closed downward, namely $xy \in T$ implies $x \in T$, for $x, y \in \mathbb{N}^*$. We view these special trees as structures of the signature E , where $E(x, y)$ if and only if $y = xz$ for some $z \in \mathbb{N}$. Thus, for every $x \in \mathbb{N}^*$ the set $\{y \mid E(x, y)\}$ can be thought as the set of all *immediate successors* of x .

A special tree T is *recursively enumerable* if the set T is the domain of the function computed by a Turing machine. We will use the following fact from computable model theory [8, Thm 3.2].

Lemma 5.4 *The isomorphism problem for recursively enumerable special trees is Σ_1^1 -complete. In fact, the trees can be chosen to be subtrees of $\{2n : n \in \mathbb{N}\}^*$.*

It is clear from the proof in [8] that the trees obtained are special (namely, subtrees of \mathbb{N}^*). By a mere change of notation one obtains subtrees of $\{2n : n \in \mathbb{N}\}^*$.

Lemma 5.5 *The special tree \mathbb{N}^* has an automatic presentation \mathcal{A}_1 .*

Proof. Consider the prefix relation \leq_p on the set of all binary strings. The tree \mathbb{N}^* is isomorphic to the automatic tree $\mathcal{A}_1 = (\{0, 1\}^* 1 \cup \{\lambda\}; E_1)$, where E_1 is the set of all pairs (x, y) such that y is the immediate \leq_p -successor of x . The isomorphism is established via the computable mapping sending $n_1 \dots n_k$ to $0^{n_1} 1 \dots 0^{n_k} 1$ and the root to λ .

Define $\mathcal{S} = \{0, 1\}^* 1 \cup \{\lambda\}$. From now on we make the following conventions:

1. All special trees we consider will be viewed as subsets of (\mathcal{S}, \leq_p) .
2. The set of inputs for all Turing machines considered are strings from \mathcal{S} . Each $w \in \mathcal{S}$ is identified with the initial configuration starting from w .
3. All Turing machines considered are reversible.
4. The domains of all Turing machines considered are assumed to be downward closed. Thus, the domains of all Turing machines form recursively enumerable special trees.

Our goal is to transform every recursively enumerable tree $T \subset \mathcal{S}$ into an automatic structure \mathcal{A}_T , where T is the reversible machine with domain T . The idea is to attach computations of T to the initial configurations in \mathcal{S} . For all recursively enumerable trees T_1 and T_2 , T_1 and T_2 are isomorphic if and only if the automatic structures \mathcal{A}_{T_1} and \mathcal{A}_{T_2} are isomorphic. To ensure this we also have to attach infinitely many chains of

each finite length to the initial configurations, for in that case the length of a halting computation does not make a difference.

Let \mathcal{R} be a reversible Turing machine. A **chain** is an initial segment of (\mathbb{N}, S) . Let \mathcal{J} be the graph consisting of infinitely many finite chains of every finite length. We denote by J the set of vertices of \mathcal{J} , and the bases of the chains in \mathcal{J} by b_1, b_2, \dots . The following is not hard to prove:

Lemma 5.6 *The graph \mathcal{J} has an automatic presentation.*

To construct $\mathcal{A}_\mathcal{R}$, with every $w \in A_1$ associate a graph \mathcal{J}_w defined as follows. The vertex set of \mathcal{J}_w consists of all $\{(w, j) \mid j \in J\}$ and edges between (w, j) and (w, j') if and only if (j, j') is an edge in \mathcal{J} . Put connecting edges from w into each (w, b_i) . Let \mathcal{A}_2 be the graph consisting of \mathcal{A}_1 and every \mathcal{J}_w and the connecting edges. Clearly, the graph \mathcal{A}_2 is automatic.

To the set \mathcal{A}_2 add all the configurations of the Turing machine \mathcal{R} and all the edges of the configuration space of \mathcal{R} . Note that each $w \in \mathcal{S}$, which is an initial configuration of \mathcal{R} is by Convention 2 already in \mathcal{A}_2 . In addition, add a disjoint automatic copy of the graph \mathcal{J} and an automatic copy of infinitely many infinite chains. Call all chains added at this stage **isolated chains**. The resulting graph is denoted by $\mathcal{A}_\mathcal{R}$.

The proof of the following is straightforward and is omitted.

Lemma 5.7 *The graph $\mathcal{A}_\mathcal{R}$ is automatic.*

We summarise the structure of the graph $\mathcal{A}_\mathcal{R}$. Firstly, in $\mathcal{A}_\mathcal{R}$ there are infinitely many isolated chains of every (finite and infinite) length. With each element w in \mathcal{A}_1 there is an associated structure \mathcal{J}_w , which consists of infinitely many chains of every finite length, and no infinite chain. Finally with every initial configuration w (which is an element of $\mathcal{A}_\mathcal{R}$ and belongs to the infinitely branching tree \mathcal{A}_1) there is a chain with base w that is the computation path from the configuration space of Turing machine T . These observations prove the following.

Lemma 5.8 *Let w be an initial configuration of $\mathcal{A}_\mathcal{R}$. Then w is the base of infinitely many chains of every finite length. Also the Turing machine T halts on w if and only if there is no infinite chain with base w . In case T does not halt on w there is exactly one infinite chain with base w .*

Lemma 5.9 Let T_1 and T_2 be computable trees which are domains of reversible Turing machines \mathcal{R}_1 and \mathcal{R}_2 respectively. Then the trees T_1 and T_2 are isomorphic if and only if the automatic graphs $\mathcal{A}_{\mathcal{R}_1}$ and $\mathcal{A}_{\mathcal{R}_2}$ are isomorphic.

Proof First note that the domain T of \mathcal{R} consists of all w in $\mathcal{A}_{\mathcal{R}}$ that are initial configurations and are not the base of an infinite chain. Hence T may be thought of as a subgraph of $\mathcal{A}_{\mathcal{R}}$. Now suppose $\mathcal{A}_{\mathcal{R}_1}$ is isomorphic to $\mathcal{A}_{\mathcal{R}_2}$ via the map ϕ . Note that w is the base of an infinite chain if and only if $\phi(w)$ is the base of an infinite chain. Hence ϕ is an isomorphism between the trees T_1 and T_2 viewed as subgraphs of $\mathcal{A}_{\mathcal{R}_1}$ and $\mathcal{A}_{\mathcal{R}_2}$ respectively.

Conversely assume T_1 and T_2 are isomorphic via ψ (as before we may assume that T_i is a subgraph of $\mathcal{A}_{\mathcal{R}_i}$). Consider the sets I_1 and I_2 of all isolated chains in $\mathcal{A}_{\mathcal{R}_1}$ and $\mathcal{A}_{\mathcal{R}_2}$. There is an isomorphism between I_1 and I_2 , since both consist of infinitely many chains of every length (finite and infinite), independently of the chains of the configuration graphs that are *not* defined by initial configurations. Therefore it suffices to establish an isomorphism extending ψ on the rest of the structure.

If $w \in T_1$ then there is an isomorphism between all the chains with base w and all the chains with base $\psi(w)$. Indeed each is the base of infinitely many chains of every finite length and no chain of infinite length, independently of the (possibly different) lengths of the finite computations of R_1 on w and R_2 on $\psi(w)$. Hence extend ψ from the chains defined by elements of T_1 to the chains defined by elements of T_2 .

Suppose $w \in T_1$ and consider the set S_1 of immediate successors of w that are initial configurations but not in T_1 . Similarly write S_2 for the set of immediate successors of $\psi(w)$ that are initial configurations but not in T_2 . By the extra condition in Lemma 5.4 that the trees be subtrees of $\{2n : n \in \mathbb{N}\}^*$, both S_1 and S_2 are infinite. So we may extend ψ to those immediate successors by adding a bijection between S_1 and S_2 .

Finally, for any initial configurations $v_1 \in S - T_1$ and $v_2 \in S - T_2$, there is an isomorphism between the nodes in S extending v_1 and the ones extending v_2 . This isomorphism extends to the attached chains (as there is one infinite chain and infinitely many of each finite length). So we may extend ψ to an isomorphism of $\mathcal{A}_{\mathcal{R}_1}$ and $\mathcal{A}_{\mathcal{R}_2}$. \square

Hence we have reduced the isomorphism problem for recursively enumerable trees to the isomorphism problem for automatic graphs. The main result now follows.

Theorem 5.10 The isomorphism problem for automatic structures is Σ_1^1 -complete.

References

- [1] M. Benedikt, L. Libkin, T. Schwentick, and L. Segoufin. A Model-Theoretic approach to regular string relations. In *Proceedings of the 16th Annual IEEE Symposium on Logic in Computer Science* (LICS 2001), pages 431–440, June 16–19 2001.
- [2] C. H. Bennett. Logical reversibility of computation. *IBM Journal of Research and Development*, 17(6):525–532, November 1973.
- [3] A. Blumensath. *Automatic Structures*. Diploma thesis, RWTH Aachen, 1999.
- [4] A. Blumensath and E. Grädel. Automatic structures. In *15th Symposium on Logic in Computer Science* (LICS 2000), pages 51–62, June 2000.
- [5] C. Delhomme. Non-automaticity of ω^ω , 2001. manuscript.
- [6] C. Delhomme. The rado graph is not automatic, 2001. manuscript.
- [7] S. Eilenberg, C.C. Elgot, and J.C. Shepherdson. Sets recognised by n -tape automata. *Journal of Algebra*, 13(4):447–464, 1969.
- [8] S.S. Goncharov and J.F. Knight. Computable structure and non-structure theorems. *Algebra and Logic*, 41(6):351–373, 2002.
- [9] W. A. Hodges. *Model Theory*. Cambridge University Press, 1993.
- [10] H. Ishihara, B. Khoussainov, and S. Rubin. Some results on automatic structures. In *17th Symposium on Logic in Computer Science* (LICS 2002), pages 235–242, July 2002.
- [11] B. Khoussainov and A. Nerode. Automatic presentations of structures. *Lecture Notes in Computer Science*, 960:367–392, 1995.
- [12] B. Khoussainov, S. Rubin, and F. Stephan. Automatic linear orders and trees, 2003. CDMTCS technical report 208, Department of Computer Science, University of Auckland.
- [13] S. Rubin. *Automatic Structures*. Phd thesis, University of Auckland, 2004. In preparation.

Definability and Regularity in Automatic Structures

Bakhadyr Khoussainov¹, Sasha Rubin², and Frank Stephan^{3*}

¹ Computer Science Department, The University of Auckland, New Zealand
`bmk@cs.auckland.ac.nz`

² Mathematics Department, The University of Auckland, New Zealand
`rubin@math.auckland.ac.nz`

³ National ICT Australia, Sydney Research Laboratory at Kensington, Australia
`fstephan@cse.unsw.edu.au`

Abstract. An automatic structure \mathcal{A} is one whose domain A and atomic relations are finite automaton (FA) recognisable. A structure isomorphic to \mathcal{A} is called automatically presentable. Suppose R is an FA recognisable relation on A . This paper concerns questions of the following type. For which automatic presentations of \mathcal{A} is (the image of) R also FA recognisable? To this end we say that a relation R is *intrinsically regular* in a structure \mathcal{A} if it is FA recognisable in every automatic presentation of the structure. For example, in every automatic structure all relations definable in first order logic are intrinsically regular. We characterise the intrinsically regular relations of some automatic fragments of arithmetic in the first order logic extended with quantifiers \exists^∞ interpreted as ‘there exists infinitely many’, and $\exists^{(i)}$ interpreted as ‘there exists a multiple of i many’.

1 Introduction

This paper investigates the relationship between regularity, that is FA recognisability, and definability in automatic structures. Roots of this topic go back to the results of Büchi and Elgot in the 1960’s who proved the equivalence between regularity and weak monadic second order logic. A recasting of this result says that (the coding of) a relation is regular if and only if the relation it is first order definable in the structure $(\mathbb{N}, +, |_k)$, where $+$ is the addition and $n|_k m$ means that n is a power of k and n divides m . Intimately related is the work of Cobham, Semenov, Muchnik, Bruyére and others that investigates the relationship between regular relations of (coded) natural numbers and definability in certain fragments of arithmetic; see [2] for a good exposition. This paper continues and complements these lines of research by initiating the study of the relationship

* National ICT Australia is funded by the Australian Government’s Department of Communications, Information Technology and the Arts and the Australian Research Council through Backing Australia’s Ability and the ICT Centre of Excellence Program.

between regularity and definability in the general setting of arbitrary automatic structures.

Assume one has a structure \mathcal{A} that can be described by means of finite automata. This is formalised in Definition 2.3 that says that there is an encoding of the elements of the structure under which the domain A of the structure and its atomic relations are all regular. Such a structure is called *automatic*. In this case we say that the coded structure is an *automatic presentation* of \mathcal{A} . Automatic presentations of \mathcal{A} can be regarded as finite automata implementations of the structure \mathcal{A} . For instance, if $k > 1$, then a least-significant-digit-first base k encoding of the natural numbers gives rise to automatic presentations of (\mathbb{N}, S) , (\mathbb{N}, \leq) , $(\mathbb{N}, +)$ and $(\mathbb{N}, +, |_k)$. Now assume that $R \subset A^m$ is a relation, not necessarily in the language of \mathcal{A} . For example, R may be the reachability relation if \mathcal{A} is a graph, or R may be the dependency relation if \mathcal{A} is a group. It may well be the case that in one automatic presentation of \mathcal{A} the relation R is recognised by a finite automaton, and in another automatic presentation it is not. Thus, automata-theoretic properties of the relation R are dependent on the automata that describe \mathcal{A} . Our goal is to study those relations in \mathcal{A} that are regular under *all* automatic presentations of \mathcal{A} , and to understand which structures ensure a relation is regular in all automatic presentations and which do not. Formally, we introduce the following definition:

Definition 1.1. See [1]. *A relation R is intrinsically regular in an automatic structure \mathcal{A} if for every automatic structure \mathcal{B} isomorphic to \mathcal{A} the image of the relation R in \mathcal{B} is regular. Denote by $IR(\mathcal{A})$ the set of intrinsically regular relations in \mathcal{A} .*

Thus the intrinsically regular relations in \mathcal{A} are those for which regularity is invariant under all automatic presentations of \mathcal{A} . A natural class of intrinsically regular relations is the class of relations definable in the first order logic. We now single out this class of relations in the following definition:

Definition 1.2. *A relation R is first order (FO) definable in \mathcal{A} if there exists a first order formula $\phi(\bar{x}, \bar{c})$ with parameters \bar{c} from \mathcal{A} such that $R = \{\bar{a} \mid \mathcal{A} \models \phi(\bar{a}, \bar{c})\}$. Denote by $FO(\mathcal{A})$ the set of all first order definable relations in \mathcal{A} .*

A fundamental result of automatic structures is stated as follows.

Fact 1.3. *Let \mathcal{A} be an automatic structure. There exists an algorithm that from a FO definition ϕ in \mathcal{A} of a relation R produces an automaton recognising R . In particular, $FO(\mathcal{A}) \subset IR(\mathcal{A})$.*

A proof may be found in [6] or [3]; in this paper we will use this fact without explicitly referencing it. Naturally, one asks whether the converse holds. It turns out that although this is sufficient for some structures, for instance $(\mathbb{N}, +)$ and $(\mathbb{N}, +, |_m)$, in general it is not.

Extend the FO predicate logic with quantifiers \exists^∞ and $\exists^{(i)}$, where $i \in \mathbb{N}$, whose interpretations are as follows. The formula $\exists^\infty x \phi(x)$ means there are infinitely many x such that $\phi(x)$ holds, and the formula $\exists^{(i)} x \phi(x)$ means that there are exactly n elements x such that $\phi(x)$ holds and n is a multiple of i . Denote the logic by $FO^{\infty, \text{mod}}$. Say that a relation R is $FO^{\infty, \text{mod}}$ definable in a

structure \mathcal{A} if there is a $\text{FO}^{\infty, \text{mod}}$ -formula $\phi(\bar{x}, \bar{a})$, where \bar{a} is a finite tuple of elements, such that $R = \{\bar{c} \mid \mathcal{A} \models \phi(\bar{c}, \bar{a})\}$. Denote by $\text{FO}^{\infty, \text{mod}}(\mathcal{A})$ the set of relations that are $\text{FO}^{\infty, \text{mod}}$ definable in \mathcal{A} . Then Fact 1.3 can be extended as follows.

Theorem 3.2. See [3].¹ *Let \mathcal{A} be an automatic structure. There exists an algorithm that from a $\text{FO}^{\infty, \text{mod}}$ definition ϕ of a relation R produces an automaton recognising R . In particular,*

$$\text{FO}^{\infty, \text{mod}}(\mathcal{A}) \subset \text{IR}(\mathcal{A}).$$

Consequently, there is a neat characterisation of the intrinsically regular relations of (\mathbb{N}, \leq) in terms of $\text{FO}^{\infty, \text{mod}}$:

Theorem 3.3.

$$\text{IR}(\mathbb{N}, \leq) = \text{FO}^{\infty, \text{mod}}(\mathbb{N}, \leq) = \text{FO}(\mathbb{N}, \leq, M^2, M^3, \dots).$$

In order to show that a particular relation is intrinsically regular in a given automatic structure, one needs to provide a mechanism for extracting an automaton recognising the relation from automatic presentations of the structure. A perfect illustration of this is the subset like construction proof of Theorem 3.2. In order to show that a particular relation is not intrinsically regular, one needs to construct automata that, on the one hand, present the structure; and on the other, preclude the existence of automata recognising the given relation. The following theorem shows that the unary relations M^k are not intrinsically regular for the structure (\mathbb{N}, S) .

Theorem 4.1. *For every $k \geq 2$, there is an automatic presentation of (\mathbb{N}, S) in which the image of the set M^k is not regular.*

Consequently we have the following partial result.

Corollary 4.2. For $R \subset \mathbb{N}$,

$$R \in \text{IR}(\mathbb{N}, S) \text{ if and only if } R \in \text{FO}^{\infty, \text{mod}}(\mathbb{N}, S).$$

Theorem 4.1 and its proof may be applied to construct automatic structures with pathological properties. The first application is concerned with the reachability problem in automatic graphs. It is known that the reachability problem for automatic graphs is not decidable, see [3]. The underlying reason for this is that such automatic graphs necessarily have infinitely many components. In fact, the reachability problem is decidable if the given graph is automatic and has finitely many components. A natural question is whether or not the reachability relation for automatic graphs with finitely many components can be recognised by finite automata. This is answered in the following corollary:

Corollary 4.3. *There exists an automatic presentation of a graph with exactly two connected components each isomorphic to (\mathbb{N}, S) in which the reachability relation is not regular.*

¹ In a personal communication with the first author, A. Blumensath has mentioned having obtained this result.

The second application of Theorem 4.1 is on the structure (\mathbb{Z}, S) , where \mathbb{Z} is the set of all integers and S is the successor function. A *cut* is a set of the form $\{x \in \mathbb{Z} \mid x \geq n\}$, where $n \in \mathbb{Z}$ is fixed. In all previously known automatic presentations of (\mathbb{Z}, S) each cut is a regular set. The corollary below states the existence of a counterexample:

Corollary 4.4. *There exists an automatic presentation of (\mathbb{Z}, S) in which no cut is regular.*

Finally, we mention that one of the central topic in modern computable model theory, first initiated by Ash and Nerode in [1], is concerned with understanding the relationship between definability and computability, see [5, Chapter 3] for the current state of the area. For a computable structure \mathcal{A} , that is one whose atomic diagram is a computable set, a relation R is *intrinsically recursively enumerable* if in all computable isomorphic copies of \mathcal{A} the relation R is recursively enumerable. In [1] Ash and Nerode show that under some natural conditions put on \mathcal{A} , the relation R is intrinsically recursively enumerable if and only if it is definable as an effective disjunction of existential formulas. One may therefore regard the topic of this paper as a refined version of the Ash-Nerode program in which the class of automatic structures is considered rather than the class of all computable structures.

Question 1.4. *Characterise the intrinsically regular relations in \mathcal{A} as those definable in a suitable logic of \mathcal{A} .*

The results of this paper suggest that the logic is $\text{FO}^{\infty, \text{mod}}$.

The rest of the paper is organised as follows. The next section contains automata preliminaries including the definition of an automatic structure and a description of simple properties of intrinsically regular relations. The remaining sections contain proofs of some of the results stated in the introduction. Due to space constraints some of the proofs are replaced by sketches or completely omitted. The complete proofs can be found in the full version of this paper which is available as a technical report of the Centre for Discrete Mathematics and Theoretical Computer Science in Auckland.

2 Automata Preliminaries

A thorough introduction to automatic structures can be found in [3] and [6]. In this section, familiarity with the basics of finite automata theory is assumed though for completeness and to fix notations, the necessary definitions are included here. A *finite automaton* \mathcal{A} over an alphabet Σ is a tuple (S, ι, Δ, F) , where S is a finite set of *states*, $\iota \in S$ is the *initial state*, $\Delta \subset S \times \Sigma \times S$ is the *transition table* and $F \subset S$ is the set of *final states*. A *computation* of \mathcal{A} on a word $\sigma_1 \sigma_2 \dots \sigma_n$ ($\sigma_i \in \Sigma$) is a sequence of states, say q_0, q_1, \dots, q_n , such that $q_0 = \iota$ and $(q_i, \sigma_{i+1}, q_{i+1}) \in \Delta$ for all $i \in \{0, 1, \dots, n-1\}$. If $q_n \in F$, then the computation is *successful* and we say that automaton \mathcal{A} *accepts* the word. The *language* accepted by the automaton \mathcal{A} is the set of all words accepted by \mathcal{A} . In

general, $D \subset \Sigma^*$ is *finite automaton recognisable*, or *regular*, if D is the language accepted by a finite automaton \mathcal{A} .

Classically finite automata recognise sets of words. The following definitions extends recognisability to relations of arity n , called *synchronous n -tape automata*. Informally a synchronous n -tape automaton can be thought of as a one-way Turing machine with n input tapes. Each tape is regarded as semi-infinite having written on it a word in the alphabet Σ followed by an infinite succession of blanks, \diamond symbols. The automaton starts in the initial state, reads simultaneously the first symbol of each tape, changes state, reads simultaneously the second symbol of each tape, changes state, etc., until it reads a blank on each tape. The automaton then stops and accepts the n -tuple of words if it is in a final state. The set of all n -tuples accepted by the automaton is the relation recognised by the automaton. Here is a formalization. Let Σ_\diamond be $\Sigma \cup \{\diamond\}$, where $\diamond \notin \Sigma$.

Definition 2.1. Write Σ_\diamond for $\Sigma \cup \{\diamond\}$ where \diamond is a symbol not in Σ . The convolution of a tuple $(w_1, \dots, w_n) \in \Sigma^{*n}$ is the string $\otimes(w_1, \dots, w_n)$ of length $\max_i |w_i|$ over alphabet $(\Sigma_\diamond)^n$ defined as follows. Its k 'th symbol is $(\sigma_1, \dots, \sigma_n)$ where σ_i is the k 'th symbol of w_i if $k \leq |w_i|$ and \diamond otherwise.

The convolution of a relation $R \subset \Sigma^{*n}$ is the relation $\otimes R \subset (\Sigma_\diamond)^{n*}$ formed as the set of convolutions of all the tuples in R . That is $\otimes R = \{\otimes w \mid w \in R\}$.

Definition 2.2. An n -tape automaton on Σ is a finite automaton over the alphabet $(\Sigma_\diamond)^n$. An n -ary relation $R \subset \Sigma^{*n}$ is finite automaton recognisable or regular if its convolution $\otimes R$ is recognisable by an n -tape automaton.

We now relate n -tape automata to structures. A *structure* \mathcal{A} consists of a set A called the *domain* and some relations and operations on A . We may assume that \mathcal{A} only contains relational predicates as the operations can be replaced with their graphs. We write $\mathcal{A} = (A, R_1^A, \dots, R_k^A, \dots)$ where R_i^A is an n_i -ary relation on A . The relation R_i are sometimes called basic or atomic relations. We assume that the function $i \rightarrow n_i$ is always a computable one.

Definition 2.3. A structure \mathcal{A} is *automatic* over Σ if its domain $A \subset \Sigma^*$ is finite automaton recognisable, and there is an algorithm that for each i produces a finite automaton recognising the relation $R_i^A \subset \Sigma^{*n_i}$. An *isomorphism* from a structure \mathcal{B} to an automatic structure \mathcal{A} is an automatic presentation of \mathcal{B} in which case \mathcal{B} is called automatically presentable (over Σ). A structure is called *automatic* if it is automatic over some alphabet.

Consider the word structure $(\{0, 1\}^*, L, R, E, \preceq)$, where for all strings $x, y \in \{0, 1\}^*$ we have $L(x) = x0$, $R(x) = x1$, $E(x, y)$ iff $|x| = |y|$, and \preceq is the lexicographical order. It is automatic over Σ . The configuration graphs of Turing machines are examples of automatic structures. Write \mathbb{N} for the set of natural numbers including 0. Examples of automatically presentable structures are $(\mathbb{N}, +)$, (\mathbb{N}, \leq) , (\mathbb{N}, S) , the group $(\mathbb{Z}, +)$, the order on the rationals (\mathbb{Q}, \leq) , and the Boolean algebra of finite or co-finite subsets of \mathbb{N} .

Thus an automatic structure is one that is explicitly given by finite automata that recognise the domain and the basic relations of the structure. An automatically presentable structure is one that is isomorphic to some automatic

structure. Informally, automatically presentable structures are those that have finite automata implementations. The same structure may have different (indeed infinitely many) automatic presentations. One of our goals is to understand the relationships between different automatic presentations of a given structure and understand how the automata-theoretic properties of relations of this structure change when one varies its automatic presentation. We illustrate the introduced concepts with two examples. The first concerns the standard model of Presburger Arithmetic $(\mathbb{N}, +)$.

Example 2.4. For each $m > 1$ consider the presentation \mathcal{A}_m of \mathbb{N} over the alphabet $\Sigma_m = \{0, \dots, m-1\}$. Here the natural number $n \in \mathbb{N}$ is represented in A_m as its shortest the least-significant-digit-first base m -representation. The structure $(A_m, +_m)$ is automatic and is isomorphic to $(\mathbb{N}, +)$. Hence, these are automatic presentations of $(\mathbb{N}, +)$. Take any n -ary relation R in \mathbb{N} . Assume that R is intrinsically regular. Then the image $R^{(m)}$ of R in $(A_m, +_m)$ is regular. The well-known Cobham-Semenov Theorem, see [2], states that if both $R^{(i)}$ and $R^{(j)}$ are regular for multiplicatively independent i and j , then R is definable in $(\mathbb{N}, +)$. Thus $\text{IR}(\mathbb{N}, +) = \text{FO}(\mathbb{N}, +)$.

The second example concerns an extension of $(\mathbb{N}, +)$.

Example 2.5. Let $|_m$, for $m \geq 2$, be the binary relation where $x|_m y$ if and only if x is a power of m and x divides y . Then the structure $(\mathbb{N}, +, |_m)$ has an automatic presentation $\mathcal{A}_m = (A_m, +_m, D_m)$. So if $R \subset \mathbb{N}^m$ is intrinsically regular for $(\mathbb{N}, +, |_m)$, then its image $R^{(m)}$ is regular in \mathcal{A}_m . But a central result of automatic structures is that first order definability in the structure \mathcal{A}_m is equivalent to FA recognisability, see for instance [6]. Hence $R^{(m)}$ is first order definable in \mathcal{A}_m , and so R is first order definable in $(\mathbb{N}, +, |_m)$. Thus $\text{IR}(\mathbb{N}, +, |_m) = \text{FO}(\mathbb{N}, +, |_m)$.

3 Intrinsically Regular Relations in (\mathbb{N}, \leq)

The linearly ordered set (\mathbb{N}, \leq) has automatic presentations. For example, automatic presentations of $(\mathbb{N}, +)$ are also automatic presentations of (\mathbb{N}, \leq) . In this section we study intrinsically regular relations of this structure. Somewhat surprisingly we exhibit intrinsically regular relations of the structure (\mathbb{N}, \leq) that are not definable. We remind the reader that the only first order definable unary relations of (\mathbb{N}, \leq) are finite or co-finite, [4, Theorem 32A].

Let $M^i \subseteq \mathbb{N}$ be the set of all positions in \mathbb{N} that are multiples of i . Then these sets are not definable in (\mathbb{N}, \leq) , but are intrinsically regular:

Theorem 3.1. *For every i the unary predicate M^i is intrinsically regular in the structure (\mathbb{N}, \leq) .*

Proof. Let (D, \leq_D) be an automatic presentation of (\mathbb{N}, \leq) over Σ . We prove the case when $i = 2$; the case when $i \geq 3$ can be proved in a similar way. Let $E \subseteq D$ be the set of words corresponding to the set of all even natural numbers. Then $x \in E$ iff $\{y \in D \mid y \leq_D x\}$ has odd cardinality. Our goal is to define an automaton over Σ that accepts all such strings x . A rough idea is that the new

automaton we want to build calculates the parity of the number of paths in \mathcal{A} with second component fixed at x and accepts x when the parity of the number of successful paths is odd.

Let $\mathcal{A} = (Q_A, \iota_A, \Delta_A, F_A)$ be the automaton over Σ recognising \leq_D . We assume that the automaton \mathcal{A} is deterministic. Also, note that since the set $\{y \in D \mid y \leq_D x\}$ is finite for any string $x \in D$, we may assume the following. For each state $s \in Q_A$ there are finitely many strings of the form (v, \diamond^m) that transform the state s into a final state.

Fix a string $x \in D$ and a prefix w of x . For a state $s \in Q_A$ consider all strings v such that $|v| = |w|$ and the automaton \mathcal{A} transforms the string (v, w) to state s from the initial state ι_A . Call these strings (w, s) -strings.

The idea in constructing the desired automaton is this. We use the automaton \mathcal{A} . Processing the initial prefix w of x , for each state s , we count the parity of the number of (w, s) -strings. We keep a record of only those states s such that the number of (w, s) -strings is odd. By the time we finish processing the string x we have a record of all states s_1, \dots, s_k such that for each state s_i there are an odd number of (x, s_i) -strings. For each s_i we count the number n_i of strings of the type (\diamond^m) , with $m = |v|$, such that the string (\diamond^m) transforms s_i into a final state of \mathcal{A} . Then whether or not $x \in E$ can be decided based upon the parity of the numbers n_1, \dots, n_k . Here is a formal description of the desired automaton $\mathcal{B} = (Q_B, \iota_B, \Delta_B, F_B)$ over Σ_\diamond :

1. The set Q_B of states of \mathcal{B} are all subsets of Q_A .
2. The initial state ι_B of \mathcal{B} is $\{\iota_A\}$.
3. $\Delta_B(X, \sigma) = Y$, where Y consists of all states $s \in Q_A$ such that there are an odd number of pairs (s', σ') for which $s = \Delta_A(s', (\sigma'))$, $s' \in X$ and $\sigma' \in \Sigma_\diamond$. (Note that Y could be empty).
4. The set of final states F_B is defined as follows. Assume $X = \{s_1, \dots, s_k\}$ is a subset of Q_A . For each s_i , count the number n_i of all strings of the type (\diamond^m) , with $v \in \Sigma^*$, $m = |v|$, such that the string (\diamond^m) transforms s_i into a final state of \mathcal{A} . Then $X \in F_B$ if and only if $X \neq \emptyset$ and the number $n_1 + \dots + n_k$ is odd.

Let $x = \sigma_0 \dots \sigma_n$ be an input string for \mathcal{B} . Let m be the cardinality of the set $\{y \mid y \leq_D x\}$. Let $X_0 = \{\iota_A\}$, X_1, \dots, X_{n+1} be a run of \mathcal{B} on x . The automaton has the following property $(*)$ that can be proved by induction on $i \geq 1$:

$(*)$ A state s is in X_i if and only if the number of $(\sigma_0 \dots \sigma_{i-1}, s)$ -strings is odd. Let $X_{n+1} = \{s_1, \dots, s_n\}$. For each $s_i \in X_{n+1}$ the number of (s_i, x) -strings is odd. Consider the number n_i of all strings of the type (\diamond^m) , with $m = |v|$, such that the string (\diamond^m) transforms s_i into a final state of \mathcal{A} . From the definition of final states for \mathcal{B} , and the inductive assumption on X_n , we see that the cardinality of the set $\{y \mid y \leq_D x\}$ is odd if and only if X_n is non-empty and the number $n_1 + n_2 + \dots + n_k$ is odd. The theorem is proved. \square

As mentioned in the introduction, this proof can be generalised as follows.

Theorem 3.2. *Let \mathcal{A} be an automatic structure. There exists an algorithm that from any $FO^{\infty, \text{mod}}$ definition ϕ of a relation R produces an automaton recognising R . In particular, $FO^{\infty, \text{mod}}(\mathcal{A}) \subset IR(\mathcal{A})$.*

Proof (sketch). Constructing an automata that recognises relation definable by $\exists^{(i)} y \psi(\bar{x}, y)$ formula is done in a style similar to the proof of Theorem 3.1. Now note that $\exists^\infty y \psi(\bar{x}, y)$ is equivalent to $\forall z \exists y(y \preceq z \& \phi(\bar{x}, y))$, where \preceq is the length-lexicographic ordering on the domain of \mathcal{A} . \square

Theorem 3.3. $IR(\mathbb{N}, \leq) = FO^{\infty, \text{mod}}(\mathbb{N}, \leq) = FO(\mathbb{N}, \leq, M^2, M^3, \dots)$.

Proof (sketch). By Theorem 3.2, $FO^{\infty, \text{mod}}(\mathbb{N}, \leq) \subset IR(\mathbb{N}, \leq)$. So suppose that a relation R is intrinsically regular for (\mathbb{N}, \leq) . Then R is regular in every automatic presentation of (\mathbb{N}, \leq) . Consider the structure $(1^*, \leq)$ where \leq stands for the ordering induced by the one on the length: $1^n \leq 1^m$ whenever $n \leq m$. This structure is a (unary) automatic presentation of (\mathbb{N}, \leq) and hence the image of R in this presentation is regular. It is shown in [3] that the regular relations over the unary alphabet coincide with those that are first order definable in structure $(\mathbb{N}, \leq, M^2, M^3, \dots)$. This can be done, for example, via an analysis of finite automata recognising relations over the unary alphabet. Finally, suppose R is first order definable in $(\mathbb{N}, \leq, M^2, M^3, \dots)$. Since the M^i are $FO^{\infty, \text{mod}}$ definable in (\mathbb{N}, \leq) , then so is R . \square

We end the section with a simple application of Theorem 3.2 which gives a generalization of Theorem 3.1. A *tree* $\mathcal{T} = (T, \preceq)$ is a partially ordered set with a least element (the root) and for which every set of the form $\{y \in T \mid y \preceq x\}$ is a finite linear order. The *level* n of \mathcal{T} is the set of all $x \in T$ such that the cardinality of $\{y \in T \mid y \preceq x\}$ is n .

Corollary 3.4. Let (T, \preceq) be an automatic tree. Given $n \in \mathbb{N}$, the set $\{x \in T \mid x \text{ is on level } n \cdot m \text{ for some } m \in \mathbb{N}\}$ is a regular subset of T .

4 Intrinsic Regularity in (\mathbb{N}, S)

Consider the structure (\mathbb{N}, S) , where S is the successor function. Our goal is to show that in this structure, all intrinsically regular unary relations are those that are either finite or co-finite. We are also interested in providing automatic presentations of (\mathbb{N}, S) in which some familiar relations are regular and some not. Recall that the finite or co-finite subsets are the only unary relations of this structure that are first order definable, a property that easily follows from elimination of quantifiers [4, Theorem 31G]. The next theorem shows that the set M^k is not intrinsically regular relation in (\mathbb{N}, S) , and so by Theorem 3.1 neither is \leq .

Theorem 4.1. For every $k \geq 2$, there is an automatic presentation of (\mathbb{N}, S) in which the image of the set M^k is not regular.

Proof. Fix $k \geq 2$ and let $\Sigma = \{0, 1, \dots, k - 1\}$. We construct an automatic structure (Σ^*, f) isomorphic to (\mathbb{N}, S) . To do this, for any given string $x \in \Sigma^*$, we introduce the following auxiliary notations: $ep(x)$ is the string represented by bits of x at even positions; $op(x)$ is the string represented by bits of x at odd positions; n and m are the lengths of strings $ep(x)$ and $op(x)$, respectively. We may also treat $ep(x)$ and $op(x)$ as natural numbers written in least-significant-digit-first base k , and in particular perform addition on them. For example, if

$x = 0111001$ then $ep(x) = 0101$, $op(x) = 110$, $n = 4$ and $m = 3$; note that $m \leq n \leq m+1$ and $|x| = m+n$. We may regard the string x as the ordered pair of strings, written $\langle ep(x), op(x) \rangle$, and think of $op(x)$ as a parameter. Call strings x for which $ep(x) = k^{n-1}$ midpoints and strings for which $ep(x) = 0$ modulo k^n startpoints. Now we describe rules defining the function f . In brackets [[like this]] we explain the meaning of each rule if needed. We note in advance that all arithmetic is performed modulo k^n . Define an auxiliary function $\text{next}(x) = ep(x) + kop(x) + k - 1$ modulo k^n .

1. If $n \leq 2$ then $f(x)$ is the successor of x with respect to length-lexicographic ordering.
2. If $\langle \text{next}(x), op(y) \rangle$ is neither a midpoint nor a startpoint then $f(x) = y$, where $ep(y) = \text{next}(x)$ and $op(y) = op(x)$. [[This is the generic case according to which the successor of the string x , regarded as the pair $\langle ep(x), op(x) \rangle$, is $\langle \text{next}(x), op(x) \rangle$.]]
3. If $\langle \text{next}(x), op(y) \rangle$ is a midpoint then $f(x) = y$, where $|y| = |x|$, $ep(y) = ep(x) + \text{next}(\text{next}(x))$ modulo k^n and $op(y) = op(x)$. [[This case says that if adding $\text{next}(x)$ to $ep(x)$ produces a midpoint then the midpoint should be skipped. Note that $ep(y) = ep(x) + 2\text{next}(x)$.]]
4. If $\langle \text{next}(x), op(x) \rangle$ is a startpoint then $f(x) = y$, where $|y| = |x|$, $ep(y) = k^{n-1}$ and $op(y) = op(x)$. [[The successor of the endpoint is the midpoint.]]
5. If $\langle ep(x), op(x) \rangle$ is a midpoint and $op(x) < k^m - 1$ then $f(x) = y$, where $|y| = |x|$, $ep(y) = 0$ and $op(y) = op(x) + 1$ modulo k^n . [[This is the case when the parameter $op(x)$ is incremented by 1, and the string $ep(x)$ is initialized to the string consisting of n zeros.]]
6. If $\langle ep(x), op(x) \rangle$ is a midpoint and $op(x) = k^m - 1$ then $f(x) = 0^{n+m+1}$. [[This is the only case when the length of string x increases by one.]]

Now we explain how f acts. Fix $b \in \mathbb{N}$ congruent to $k - 1$ modulo k . For every $a \in \mathbb{N}$ there is a unique number $c \in \{0, 1, \dots, k^n - 1\}$ such that $a = b \cdot c$ modulo k^n . In other words, every element $c \in \{0, 1, \dots, k^n - 1\}$ appears exactly once in the sequence $0, b, 2b, 3b, \dots, (k^n - 1)b$, where elements are taken modulo k^n . Moreover, $k^{n-1}b$ equals k^{n-1} modulo k^n . Hence, $k^{n-1}b$ appears in the middle of this sequence. Let us assume that x is such that $ep(x) = 0$ and let $b = kop(x) + k - 1$. Then by rules 2, 5 and 6, the function f consecutively applied $k^n - 1$ times to $\langle 0, op(x) \rangle$ produces the following sequence:

$$\begin{aligned} \langle 0, op(x) \rangle, \langle b, op(x) \rangle, \dots, \langle k^{n-1} - b, op(x) \rangle, \langle k^{n-1} + b, op(x) \rangle, \\ \dots, \langle k^n - b, op(x) \rangle, \langle k^{n-1}, op(x) \rangle. \end{aligned}$$

Note that the midpoint $\langle k^{n-1}, op(x) \rangle$ has been removed from the middle of the sequence $\langle 0, op(x) \rangle, \langle b, op(x) \rangle, \dots, \langle k^n - b, op(x) \rangle$, and placed at the end. Finally rules 3 and 4 imply that f applied to the last string v in the sequence produces the string $\langle 0, op(x) + 1 \rangle$ if $op(x) \neq k^m - 1$; otherwise $f(v) = 0^{n+m+1}$. This completes the description of f .

The function f is FA recognisable because all the rules used in the definition of f can be tested by finite automata. It can be checked that (Σ^*, f) is isomorphic to (\mathbb{N}, S) , say via mapping $\pi : \Sigma^* \rightarrow \mathbb{N}$.

Our goal is to show that the image of the set $M^k = \{x \mid x \text{ is a multiple of } k\}$ is not regular in the described automatic presentation of (\mathbb{N}, S) . For this we need to have a finer analysis of the isomorphism π from (Σ^*, f) to (\mathbb{N}, S) . Denote by x' the string $\langle 0, op(x) \rangle$. One can inductively check the following for the case that $n \geq 3$.

1. The number $\pi(x')$ is congruent to 0 modulo k for all non-empty strings x .
2. There is a unique $u \leq k^n - 1$ such that $ep(x) = u \cdot (kop(x) + k - 1)$ modulo k^n . Moreover:
 - a) If $u < k^{n-1}$ then $\pi(x) = \pi(x') + u$.
 - b) If $u > k^{n-1}$ then $\pi(x) = \pi(x') + u - 1$.
 - c) If $u = k^{n-1}$ then $\pi(x) = \pi(x') + k^n - 1$.
3. If $ep(y) = 0$ and $op(y) = op(x') + 1 \leq k^m - 1$ then $\pi(y) = \pi(x') + k^n$.

Thus, from the above it is easy to see that x is in the image of M^k iff either $u < k^{n-1}$ and u is congruent to 0 modulo k or $u \geq k^{n-1}$ and u is congruent to 1 modulo k . In order to show that the image of M^k is not regular, consider all the strings x such that n is odd, $ep(x) = 1^n$ (its numerical value is $k^{n+1} - 1$), $op(x) = 0^{m-r}1^r$ (its numerical value is $k^{r+1} - 1$, so that $kop(x) + k - 1 = k^{r+2} - 1$), and $n > r + 4$. Then under these premises for every $r \in \mathbb{N}$ the minimal $n \in \mathbb{N}$ for which $x \in \pi(M^k)$ is when $n = 2r + 5$:

Indeed, $(k^{n-1} + k^{r+2} + 1) \cdot (k^{r+2} - 1) = k^{2r+4} - k^{n-1} - 1$ modulo k^n . So under the assumption that $n = 2r + 5$, this is equal to $-1 = ep(x)$ modulo k^n . Hence $u = k^{n-1} + k^{r+2} + 1 > k^{n-1}$ and so by item 2b above conclude that $\pi(x) = \pi(x') + k^{n-1} + k^{r+1}$ and so $x \in \pi(M^k)$.

For the converse, $(k^{r+2} + 1) \cdot (k^{r+2} - 1) = k^{2r+4} - 1$ modulo k^n . Hence under the assumption that $n < 2r + 5$, this is equal to $-1 = ep(x)$ modulo k^n . Now if further $r + 3 < n - 1$, then $u = k^{r+2} + 1 < k^{n-1}$, and so by item 2a above conclude that $\pi(x) = \pi(x') + k^{r+2} + 1$ and so $x \notin \pi(M^k)$.

Now we can check that $\pi(M^k)$ is not regular. Note that in the presence of $n = 2r + 5$ the assumption that $n > r + 4$ is redundant since $n \leq r + 4$ implies that $r \leq -1$ which contradicts that $r \in \mathbb{N}$. So consider the non regular set

$$Y = \{x \in \Sigma^* \mid ep(x) = 1^n, op(x) = 0^{m-r}1^r, n = 2r + 5\}.$$

It can be defined from $\pi(M^k)$ as the set of all $x \in \Sigma^*$ such that $ep(x) = 1^n$, for some odd n , $op(x) = 0^{m-r}1^r$ for some $m, r \in \mathbb{N}$, $n > r + 4$, $x \in \pi(M^k)$ and if $r + 4 < s < n$ then $(1^s, op(x)) \notin \pi(M^k)$. But since Y is not regular, neither is $\pi(M^k)$, as required. \square

Corollary 4.2. *A unary relation $R \subset \mathbb{N}$ is intrinsically regular in (\mathbb{N}, S) if and only if it is in $FO^{\infty, \text{mod}}(\mathbb{N}, S)$.*

Proof. The reverse direction is immediate. For the forward direction it is sufficient to prove that if $R \subset \mathbb{N}$ is intrinsically regular in (\mathbb{N}, S) then it is finite or co-finite; in this case it is in $FO(\mathbb{N}, S)$ and so certainly in $FO^{\infty, \text{mod}}(\mathbb{N}, S)$. It can be proved that if R is an eventually periodic set, and if it is infinite and co-infinite, then there is some period p of R such that M_p is first order definable (\mathbb{N}, S, R) . Assuming this proceed as follows. Let $R \subset \mathbb{N}$ be intrinsically regular

in (\mathbb{N}, S) . Since $(1^*, \otimes\{(1^n, 1^{n+1}) \mid n \in \mathbb{N}\})$ is an automatic presentation of (\mathbb{N}, S) , R must be eventually periodic. If R is finite or co-finite we are done. Otherwise R is regular in every presentation of (\mathbb{N}, S) and using the fact there exists a period p of R such that M_p is first order definable in (\mathbb{N}, S, R) we get that M_p is also intrinsically regular in (\mathbb{N}, S) contradicting the previous theorem. \square

The first application of the results concerns the reachability relation in automatic graphs. The reachability problem for automatic graphs is undecidable, see [3]. The reason for this is that such automatic graphs necessarily have infinitely many components. In fact for automatic graphs with finitely many components the reachability problem is decidable. A natural question is whether or not the reachability relation for automatic graphs with finitely many components can be recognised by finite automata. To answer this question, consider the following graph $\mathcal{G} = (\{0, 1\}^*, Edge)$, where $Edge(x, y)$ if and only if $f^2(x) = y$ and f is the function defined in the proof of Theorem 4.1 for $k = 2$. The graph \mathcal{G} is automatic with exactly two infinite components each being isomorphic to (\mathbb{N}, S) . One of the components coincides with M^2 , and so neither component is regular. Hence, we have the following:

Corollary 4.3. *There exists an automatic presentation of a graph with exactly two connected components each isomorphic to (\mathbb{N}, S) for which the reachability relation is not regular.*

A final application of this theorem is on the structure (\mathbb{Z}, S) . A *cut* is a set of the form $\{x \in \mathbb{Z} \mid x \geq n\}$, where $n \in \mathbb{Z}$ is fixed.

Corollary 4.4. *There is an automatic presentation of (\mathbb{Z}, S) in which no cut is regular.*

Proof (sketch). It is sufficient to find a presentation of $(\mathbb{Z}, S, 0)$ in which $\{x \in \mathbb{Z} \mid x \geq 0\}$ is not regular since every other cut is first order definable from this one. We modify the presentation in the proof of Theorem 4.1 for $k = 2$, by considering the structure $(\{0, 1\}^*, g)$, where g is defined using the same notation as before. All arithmetic below is performed modulo 2^n .

1. If $n \leq 2$ then $g(x)$ is the length-lexicographic successor of x .
2. If $(ep(x) + 2op(x) + 1, op(x))$ is neither a midpoint nor a startpoint then $g(x) = y$ with $|x| = |y|$ and $ep(y) = ep(x) + 2op(x) + 1$ and $op(y) = op(x)$.
3. If $(ep(x) + 2op(x) + 1, op(x))$ is a midpoint, then
 - a) if $op(x) < 2^m - 1$ then $g(x) = y$ with $|x| = |y|$ and $ep(y) = 0$ and $op(y) = op(x) + 1$.
 - b) if $op(x) = 2^m - 1$ then $g(x) = y$ with $|y| = |x| + 1$ and $ep(y) = 0$ and $op(x) = 0$.
4. If $(ep(x) + 2op(x) + 1, op(x))$ is a startpoint, then
 - a) if $op(x) < 2^m - 1$ then $g(x) = y$ with $|x| = |y|$ and $ep(y) = 2^{n-1}$ and $op(y) = op(x) + 1$.
 - b) if $op(x) = 2^m - 1$ then
 - i. if $n = 3$ and $m = 2$ then $g(x) = \epsilon$. Otherwise,

- ii. if $n = m + 1$ then $g(x) = y$ with $|ep(y)| = n - 1$, $|op(y)| = m$ and $ep(y) = 2^{n-2}$ and $op(y) = 0$.
- iii. if $n = m$ then $g(x) = y$ with $|ep(y)| = n$ and $|op(y)| = m - 1$ and $ep(y) = 2^{n-1}$ and $op(y) = 0$.

Thus, $(\{0,1\}^*, g, \epsilon)$ is an automatic presentation of $(\mathbb{Z}, S, 0)$ in which the cut above 0 is exactly those x such that $u < 2^n - 1$. But if this set were regular then so is the image of M^2 in $(\{0,1\}^*, f)$. \square

Finally we mention that there is an automatic presentation of (\mathbb{N}, S) in which \leq is not regular but all the unary relations M^2, M^3, \dots are regular. This shows that regularity of each of the sets M^i and the successor relation S do not generally imply that the relation \leq is regular. Together with the previous result this theorem says that regularity of \leq is independent of whether or not sets M^i are regular. The proof is available in the full version of this paper.

Theorem 4.5. *The structure $(\mathbb{N}, S, M^2, M^3, \dots)$ has an automatic presentation in which the relation \leq is not regular.* \square

Acknowledgment. We would like to thank the referees for their suggestions to improve the presentation of the paper and for their thorough reading.

References

1. Christopher J. Ash and Anil Nerode. Intrinsically recursive relations. In *Aspects of effective algebra (Clayton, 1979)*, pages 26–41. Upside Down A Book Co. Yarra Glen, 1981.
2. Veronique Bruyère, Georges Hansel, Christian Michaux and Roger Villemaire. Logic and p-recognizable sets of integers. *Bull. Belg. Math. Soc.*, 1:191–238, 1994.
3. Achim Blumensath and Erich Grädel. *Automatic Structures*, Proceedings of 15th Symposium on Logic in Computer Science, LICS 2000.
4. Herbert B. Enderton. *A mathematical introduction to logic*. Academic Press, first edition, 1972.
5. Ershov et al. (eds). Handbook of Recursive Mathematics Volume 1. Studies in Logic and the foundations of Mathematics, Elsevier, 1998.
6. Bakhadyr Khoussainov and Anil Nerode. Automatic presentations of structures. *Lecture Notes in Computer Science*, 960:367–392, 1995.

Decidability of Term Algebras

Extending Partial Algebras

Bakhadyr Khoussainov and Sasha Rubin

Department of Computer Science, University of Auckland, New Zealand

Abstract. Let \mathcal{A} be a partial algebra on a finite signature. We say that \mathcal{A} has decidable query evaluation problem if there exists an algorithm that given a first order formula $\phi(\bar{x})$ and a tuple \bar{a} from the domain of \mathcal{A} decides whether or not $\phi(\bar{a})$ holds in \mathcal{A} . Denote by $E(\mathcal{A})$ the total algebra freely generated by \mathcal{A} . We prove that if \mathcal{A} has a decidable query evaluation problem then so does $E(\mathcal{A})$. In particular, the first order theory of $E(\mathcal{A})$ is decidable. In addition, if \mathcal{A} has elimination of quantifiers then so does $E(\mathcal{A})$ extended by finitely many definable selector functions and tester predicates. Our proof is a refinement of the quantifier elimination procedure for free term algebras. As an application we show that any finitely presented term algebra has a decidable query evaluation problem. This extends the known result that the word problem for finitely presented term algebras is decidable.

1 Introduction

The (free) algebra of terms plays an important role in many areas of computer science and algebra. It is the unique universal object that can be mapped homomorphically onto any given algebra (over a fixed signature). This provides a bijection between congruences of the term algebra and the class of all algebras (over that signature). Since finite trees can be represented as terms, the algebra of terms appears in computer science. For instance, in automata theory regular languages of trees can be identified with congruences of finite index of the algebra of terms. In logic programming, terms are used as basic objects in unification algorithms. In modern object oriented programming many data structures are stored and manipulated as terms. Other applications are in constraint databases, pattern matching, type theory and the theory of algebraic specification.

In logic and computability, the term algebra attracts much attention due to the fact that its first order theory is decidable. This was first proved by Mal'cev in [15]. His proof uses the method of elimination of quantifiers. This result has been reproved and extended by others in different settings. Rybina and Voronkov in [17] applied the method of elimination of quantifiers to show that the term algebra with queues has a decidable first order theory. Korovin and Voronkov in [8] prove that the existential fragment of the term algebra with the Knuth-Bendix ordering is decidable. Manna, Zhang and Sipma in [16] prove that the term algebra with the length function for terms has a decidable theory using the elimination of quantifiers. They also prove that the theory of the term algebra

that involves k -alternations of quantifiers, regardless of the total number of the quantifiers, is at most k -fold exponential. Vorobyov in [18] and Compton and Henson in [4] prove that the decision problem for the first order theory of the term algebra has a non-elementary lower bound.

In this paper, we extend the decidability result for term algebras to a much more general setting. A *partial algebra* (or simply an *algebra*) \mathcal{A} is a structure whose basic operations are partial functions on A . In case all the functions are total on A , we may stress this and call \mathcal{A} a *total algebra*. Partial algebras naturally occur when one restricts the basic operations of a total algebra \mathcal{A} to some $B \subset A$ that is not closed under the basic operations. In this case the value of a term in \mathcal{B} may be undefined, in which case \mathcal{B} is a partial algebra and not a total algebra.

The algebra freely generated by \mathcal{A} , called the *free total extension* of \mathcal{A} and written as $E(\mathcal{A})$, is the total algebra generated by \mathcal{A} with a sort of universal mapping property: every homomorphism from \mathcal{A} into any total algebra \mathcal{B} can be extended to a homomorphism from $E(\mathcal{A})$ into \mathcal{B} . We remark that $E(\mathcal{A})$ is a natural object in universal algebra (see [9, Section 28]) as well as in computer science. For instance in the theory of algebraic specification partial algebras are a natural way of treating errors such as division by zero (see [1]); here $E(\mathcal{A})$ are used as models of specifications. The algebra $E(\mathcal{A})$ is also used in providing non-standard models of Clarke's Axioms (see [14]). Also [11] uses $E(\mathcal{A})$ to extend the Myhill-Nerode theorem with ‘finitely generated congruence’ instead of ‘congruence of finite index’.

We say that an algebra \mathcal{A} has a *decidable query evaluation problem* if there exists an algorithm that given a first order formula $\phi(\bar{x})$ and a tuple \bar{a} from the domain of \mathcal{A} decides whether or not $\phi(\bar{a})$ holds in \mathcal{A} . In particular if \mathcal{A} has a decidable query evaluation problem then its first order theory is decidable. Our main result states:

If \mathcal{A} has a decidable query evaluation problem then so does $E(\mathcal{A})$.

In particular, the first order theory of $E(\mathcal{A})$ is decidable. In addition, if \mathcal{A} has elimination of quantifiers then so does $E(\mathcal{A})$ extended by finitely many definable selector functions and tester predicates. These results are proved by rewriting a formula Φ of $E(\mathcal{A})$ into a formula Φ' that can be evaluated in A so that $E(\mathcal{A}) \models \Phi$ if and only if $\mathcal{A} \models \Phi'$. The technique is a refinement of the quantifier elimination procedure in [16].

A *finitely presented term algebra* is the quotient of a free term algebra by finitely many ground term equations. The word problem for finitely presented term algebras is decidable [10]. As a corollary to our main result we have, we feel, a clean proof that the first order theory of a finitely presented algebra is decidable [3].

We place our result, that decidability is preserved by the free total extension of \mathcal{A} , in the realm of other constructions that preserve decidability. Direct product, disjoint union and under suitable conditions the ω -product preserve decidability of the query evaluation problem. Another example is the construction of the tree-like unfolding from [19] that preserves the monadic second order

theory. Finally we mention a construction from [12] that resembles the one in this paper. There a purely relational structure \mathcal{A} is lifted by first extending the signature by adding new function symbols from a functional signature Σ . One then considers the Σ -term algebra generated by constants from A . Finally, the relations of the algebra \mathcal{A} are lifted in a natural way to the domain of the terms. The resulting structure is called the Σ -term power of \mathcal{A} . It is proved that if \mathcal{A} has decidable first order theory then so does the Σ -term power of \mathcal{A} .

Here is a brief outline of this paper. The next section gives basic definitions, examples, and facts about structures with decidable query evaluation problem. Section 3 presents a formal definition of the free total extensions for partial algebras and some of the properties of these extensions. Section 4 is devoted to proving the main result of this paper. The final section applies the main result to show that each finitely presented term algebra has decidable query evaluation problem.

2 Structures with Decidable Query Evaluation Problem

A *structure* consists of a domain A of elements, and basic operations $f^A, g^A \dots$ on A and relations P^A, Q^A, \dots on A . The *signature* of \mathcal{A} is $(f, g, \dots, P, Q, \dots)$. We view constants as operations of arity 0. In general, the operations f^A may be partial functions.

Convention: For the purpose of this paper, the domain A is a decidable set from a decidable domain such as the strings over a finite alphabet, or ground terms over a finite ranked alphabet, or natural numbers. In particular all the structures we consider are countable.

Definition 1. The query evaluation problem for the structure \mathcal{A} is the set $QEP(\mathcal{A})$ of all pairs $(\phi(\bar{x}), \bar{a})$ such that $\mathcal{A} \models \phi(\bar{a})$ where $\phi(\bar{x})$ is a first order formula of \mathcal{A} and \bar{a} is a tuple of elements from A . If there is an algorithm deciding $QEP(\mathcal{A})$ then we say that \mathcal{A} has decidable query evaluation problem.

We will abbreviate the phrase ‘query evaluation problem’ as QEP.

In other words, \mathcal{A} has decidable QEP means that its elementary diagram with constants naming every element in A is decidable. We remark that this definition can be extended to other logics besides first order. Here are several examples of structures with decidable QEP.

Example 1. Every finite structure has decidable QEP.

Recall that a theory T is a set of sentences closed under deduction. A theory T admits effective quantifier elimination if there is an effective procedure that transforms a formula into an equivalent (in T) quantifier free formula. Say that a structure \mathcal{A} admits effective quantifier elimination if its first order theory does. Specific examples include algebraically closed fields, vector spaces over finite fields, term algebras extended with selector functions, etc (see [5] for examples).

Example 2. If \mathcal{A} admits effective elimination of quantifiers and the domain and basic operations of \mathcal{A} are decidable, then \mathcal{A} has decidable QEP.

Automatic structures are relational structures whose predicates are recognised by synchronous finite automata. For precise definitions see [7].

Example 3. Automatic structures have decidable QEP.

For issues on complexity of query evaluation problems for automatic structures see [2], [13], and for examples of automatic structures see [6], [7]. For these structures definable relations are, in fact, recognised by finite automata.

Example 4. Every decidable consistent theory T has a model with decidable QEP.

Indeed, the classical Henkin construction can be made effective; and the constructed model has decidable QEP.

Example 5. If \mathcal{A} has decidable QEP then every structure that is first order definable in \mathcal{A} also has decidable QEP.

Example 6. Let \mathcal{A} and \mathcal{B} be structures of a signature Σ with decidable QEP. Then the following structures have decidable QEP: the product $\mathcal{A} \times \mathcal{B}$ (Feferman, Vaught, 59); the disjoint union $\mathcal{A} \oplus \mathcal{B}$, where the domain of $\mathcal{A} \oplus \mathcal{B}$ is the union of A and B and for each $P \in \Sigma$ the predicate $P^{\mathcal{A} \oplus \mathcal{B}}$ is the union $P^{\mathcal{A}} \cup P^{\mathcal{B}}$ and there is a unary predicate for A .

3 Free Total Extensions of Partial Algebras

Partial Algebras: A *partial algebra*, or simply an *algebra*, is a structure $\mathcal{A} = (A, f^A, g^A, \dots, h^A)$ consisting of a domain of elements A and finitely many partial functions on A . Constants are viewed as functions of arity 0. Incase all the functions are total we call the structure a *total algebra*. The signature of \mathcal{A} is (f, g, \dots, h) and is called a *functional signature* since it does not contains symbols for relations. Note that a term, such as $f(g(a, b))$, may not have a value in A , in which case we say that *(the value of) the term is undefined in \mathcal{A}* . All structures implicitly have the symbol $=$ for equality. In a partial algebra \mathcal{A} two terms s and t are defined to be equal, written $s = t$, if they are both defined in \mathcal{A} and have the same value in \mathcal{A} (so called existential equality). Tuples of elements a_i of A and tuples of variables x_i are denoted by \bar{a} and \bar{x} respectively.

Term Algebras: Let Σ be a functional signature and C a non-empty domain. Then the set of *ground terms* over C , written $GT_{\Sigma}(C)$ or simply $GT(C)$, is defined inductively as follows:

- Every element of C is a ground term.
- If f is an n -ary symbol from Σ , and \bar{t} is an n -tuple of ground terms, then $f(\bar{t})$ is a ground term.

The *free term algebra generated by* C is $(GT(C), (f)_{f \in \Sigma})$ where the value of f on \bar{t} is defined as the ground term $f(\bar{t})$. It is a total algebra.

The *depth* of ground terms is defined as follows: terms in C have depth 0; if $f(t_1, \dots, t_k)$ is not in C , then its depth is 1 more than the maximum of the depths of the t_i .

Free Total Extensions: Let \mathcal{A} be a partial algebra on signature Σ . We define what it means to extend \mathcal{A} to a total algebra $E(\mathcal{A})$ in the free-est possible way. First we give an explicit construction and then the usual one in terms of homomorphisms.

For any ground term $t \in GT(A)$ define its *canonical form* with respect to the structure \mathcal{A} , written $t_{\mathcal{A}}^{(c)}$ or simply $t^{(c)}$, by induction as follows.

- If $t = a$ and $a \in A$ then define $t^{(c)}$ as a .
- Assume $t = f(t_1, \dots, t_n)$, and $t_1^{(c)}, t_2^{(c)}, \dots, t_n^{(c)}$ have been defined. If each $t_i^{(c)}$ is in A and the value $f(t_1^{(c)}, t_2^{(c)}, \dots, t_n^{(c)})$ is defined in \mathcal{A} and equals $b \in A$ then define $t^{(c)}$ as b . Otherwise $t^{(c)}$ is defined as $f(t_1^{(c)}, t_2^{(c)}, \dots, t_n^{(c)})$.

The *algebra of canonical terms with respect to* \mathcal{A} is the total algebra over signature Σ , whose domain is the set of canonical terms $t^{(c)}$ for $t \in GT(A)$, and for which the value of f on \bar{t} , with t_i canonical, is simply $f(\bar{t})^{(c)}$.

As we will see in a moment, this algebra is isomorphic to the free total extension of \mathcal{A} . First we need some definitions (see [9][section 13]).

Let \mathcal{C} and \mathcal{B} be partial algebras over the same signature. A *homomorphism from* \mathcal{C} *into* \mathcal{B} is a total mapping $h : \mathcal{C} \rightarrow \mathcal{B}$ so that whenever $\bar{a} \in C$, $f \in \Sigma$, and $f^{\mathcal{C}}(\bar{a})$ is defined then $f^{\mathcal{B}}(h(\bar{a}))$ is defined and is equal to $hf^{\mathcal{C}}(\bar{a})$. For $B \subset C$, say that \mathcal{C} *extends* \mathcal{B} if for every \bar{b} , $f^{\mathcal{B}}(\bar{b})$ is defined and equal to $b \in B$ if and only if $f^{\mathcal{C}}(\bar{b})$ is defined and equal to $b \in B$. Note that this allows the possibility that $f^{\mathcal{C}}(\bar{b})$ is defined (and not in B) while $f^{\mathcal{B}}(\bar{b})$ is undefined. Finally recall that a total algebra \mathcal{C} is *generated* by a set $X \subset C$ if \mathcal{C} is the smallest (under \subset) total algebra containing every total subalgebra \mathcal{B} with $X \subset B$. In this case every element of \mathcal{C} is equal to $t(\bar{b})$ for some term t and some tuple of elements \bar{b} from X .

Define a *free total extension* of \mathcal{A} (compare [11]), written $E(\mathcal{A})$, as satisfying the following properties:

1. $E(\mathcal{A})$ is a total algebra extending \mathcal{A} .
2. $E(\mathcal{A})$ is generated by the elements of A .
3. Every homomorphism h from \mathcal{A} into a total algebra \mathcal{B} can be extended to a homomorphism of $E(\mathcal{A})$ into \mathcal{B} .

Note that $E(\mathcal{A})$ is unique. Here are some examples.

Example 7. Let \mathcal{C} be the partial algebra $(C, (f)_{f \in \Sigma})$ where the f 's are undefined everywhere. Then $E(\mathcal{C})$ is the free term algebra generated by C .

Example 8. Let \mathcal{A} be a finite partial algebra. Then $E(\mathcal{A})$ is isomorphic to the quotient of $GT(A)$ by finitely many ground term equations [11].

Proposition 1. *The algebra of canonical terms with respect to \mathcal{A} is isomorphic to $E(\mathcal{A})$.*

Convention. In what follows we will work on this algebra of canonical terms directly, instead of using the abstract definition of $E(\mathcal{A})$. Moreover, in order to distinguish the original domain A of $E(\mathcal{A})$ we introduce a unary predicate A to the language.

4 Main Theorem

The main result is the following theorem.

Theorem 1. *If a partial algebra \mathcal{A} has decidable QEP then so does its free total extension $E(\mathcal{A})$.*

The proof is a mixture of the quantifier elimination procedure for free term algebras and the decision procedure of \mathcal{A} . The basic idea is to inductively remove existential quantifiers over variables specified to be outside of A .

We extend the signature $\Sigma \cup \{A\}$ of $E(\mathcal{A})$ to include new operations and relations. To avoid confusion, the operations of Σ are called *constructors*. The new operations consist of:

- unary *selector functions* f_i for every constructor f and $i \leq \text{arity}(f)$, and
- unary *tester predicates* IS_f for every constructor f .

From now on we work in this extended signature unless specified otherwise. Sequences of selector functions will be denoted by L, M, \dots and L_i, M_i, \dots for $i \in \mathbb{N}$.

Semantics: These new operations have the following semantics. Let t be a canonical term. If $t \notin A$ then there is a unique constructor f and canonical terms \bar{s} so that $t = f(\bar{s})$. In this case, define $f_i(t)$ as s_i (for $i \leq \text{arity}(f)$) and define $\text{IS}_f(t)$ as \top . Also define $g_i(t) = t$ and $\text{IS}_g(t)$ as \perp for every constructor $g \neq f$ (for $i \leq \text{arity}(g)$). On the other hand, if $t \in A$, then define $g_i(t)$ as t and $\text{IS}_g(t)$ as \perp for every constructor g (for $i \leq \text{arity}(g)$). Finally $A(t)$ holds if and only if $t \in A$. Note the selector functions f_i and the tester predicates IS_f are definable in the language of $E(\mathcal{A})$.

Terms: A term t over the extended signature of $E(\mathcal{A})$ with free variables amongst \bar{x} will be written $t(\bar{x})$. The expression

$$t[x_1/s_1(\bar{v}), \dots, x_k/s_k(\bar{v})]$$

denotes the term t where each of the mentioned x_i from \bar{x} has been replaced with the corresponding term $s_i(\bar{v})$. For instance, if $t = f(h_2(x_1), x_2)$, then $t[x_1/g(v_1)]$ is the term $f(h_2(g(v_1)), x_2)$.

Literals: To be sure, every literal in the language of $E(\mathcal{A})$ is either an equation of terms $t = s$, a disequation of terms $t \neq s$, or a tester predicate applied to a term $\text{IS}_f(t)$.

We list some basic properties of $E(\mathcal{A})$ that will be used implicitly in showing the correctness of the the algorithm provided in the next section. Here and unless specified otherwise, equivalence is in $E(\mathcal{A})$.

Lemma 1. *The algebra $E(\mathcal{A})$ satisfies the following properties:*

1. $t_1 = t_2$ implies that $t_1 \in A$ if and only if $t_2 \in A$.
2. $f(\bar{t}) \notin A \wedge f(\bar{s}) \notin A$ implies that $f(\bar{t}) = f(\bar{s})$ is equivalent to $\bigwedge t_i = s_i$.
3. $f(\bar{t}) \notin A \wedge f(\bar{s}) \notin A$ implies that $f(\bar{t}) \neq f(\bar{s})$ is equivalent to $\bigvee t_i \neq s_i$.
4. for $f \neq g$, $[f(\bar{t}) \notin A \wedge g(\bar{s}) \notin A]$ implies that $f(\bar{t}) = g(\bar{s})$ is equivalent to \perp .
5. for $f \neq g$, $[f(\bar{t}) \notin A \wedge g(\bar{s}) \notin A]$ implies that $f(\bar{t}) \neq g(\bar{s})$ is equivalent to \top .
6. $s \notin A \wedge f(\bar{t}) \notin A$ implies that the equation $s = f(\bar{t})$ is equivalent to $\bigwedge f_i s = t_i \wedge IS_f(s)$. Similarly it implies that the disequation $s \neq f(\bar{t})$ is equivalent to $\bigvee f_i s \neq t_i \vee \neg IS_f(s)$.

4.1 Quantifier Elimination

(Un)limited Quantification: An *unlimited quantification* is one of the form $Qz \notin A$ and a *limited quantification* is one of the form $Qz \in A$, where $Q \in \{\exists, \forall\}$. As a shorthand we write $Q^u z$ for unlimited quantification and $Q^l z$ for limited quantification. Also we write $\exists^u \bar{x}$ for $\exists^u x_1 \cdots \exists^u x_m$ where $\bar{x} = (x_1, \dots, x_m)$. Note that a quantification may be neither limited nor unlimited.

From now on, let \bar{x} denote existentially quantified unlimited variables and let \bar{y} denote unlimited parameters. Similarly let \bar{x}' denote quantified limited variables and let \bar{y}' denote limited parameters.

The following technical lemma shows how to remove unlimited quantification. Its proof will be the focus of this subsection.

Lemma 2. *Every formula of the form $\exists^u \bar{x} \chi(\bar{x}, \bar{y}, \bar{y}')$ where all quantifications in χ are limited, is equivalent in $E(\mathcal{A})$ to a formula of the form $\chi'(\bar{y}, \bar{y}')$, where all quantifications in χ' are also limited.*

Proof. We may assume \mathcal{A} is not a total algebra for otherwise $E(\mathcal{A})$ is isomorphic to \mathcal{A} in which case the lemma is trivial.

Consider a formula of the form

$$\exists^u \bar{x} \chi(\bar{x}, \bar{y}, \bar{y}'),$$

where all the quantifications in χ are limited. We will describe a procedure that transforms this formula into an equivalent formula where all the quantifications are limited.

The procedure will use the following techniques implicitly. We can always assume that all quantified variables are distinct by renaming if necessary.

disjunctive splitting: The process of replacing a formula of the form $\exists x(B \vee C)$ by its logical equivalent $(\exists x B) \vee (\exists x C)$.

disjunctive normal form: Every quantifier free formula can be written as $\bigvee (\bigwedge B_{i,j})$ where the $B_{i,j}$ are literals.

formula normal form: Every formula can be expressed as

$$Q_k v_k \cdots Q_1 v_1 \psi,$$

where the Q_i are blocks of quantifiers of the same type (namely \exists or \forall) and ψ is quantifier free in disjunctive normal form.

We now explain the concepts of type and type completion that will be used throughout the algorithm.

Types and Type Completions: A *type* of a term s is one of the following formulae:

- $s \in A$, or
- $s \notin A \wedge \text{IS}_g(s) \wedge \bigwedge_{f \neq g} \neg \text{IS}_f(s)$, where g is some constructor.

Note that a term has finitely many types. This definition is used in the following important concept [16].

Say that a conjunction of literals B is *type-completed* if for every subterm s in B , exactly one type of s is expressed in B . Note that a conjunction of literals can be extended to finitely many non-equivalent in $E(\mathcal{A})$ type-completed formulae. For example, a type completion of the formula $f(x) = y$ is the conjunction of $f(x) = y$ and

$$[x \in A] \wedge [f(x) \notin A \wedge \text{IS}_f(f(x)) \wedge \bigwedge_{g \neq f} \neg \text{IS}_g(f(x))] \wedge [y \notin A \wedge \text{IS}_h(y) \wedge \bigwedge_{g \neq h} \neg \text{IS}_g(y)].$$

We remark that a type completion may not be satisfiable (for instance if $f \neq h$ in the example above) Indeed in the algorithm such type completions are identified and replaced with \perp .

The *type completion* of a quantifier free formula $\phi = \bigvee \psi_i$, where each ψ_i is a conjunction of literals, is the equivalent quantifier free formula

$$\bigvee_{i,k} \psi'_{i,k},$$

where $\{\psi'_{i,k} \mid k\}$ consist of all the non-equivalent type completions of ψ_i . Here $\bigvee_{i,k} \psi'_{i,k}$ is the *type completion* of ϕ and is called *type-completed*. Note that each $\psi'_{i,k}$ is a conjunction of literals.

The *type completion* of a formula Φ is defined by the following procedure. First put Φ into formula-normal-form. So Φ is of the form

$$Q_p v_p \cdots Q_1 v_1 \psi,$$

where each Q_i is either \forall or \exists . Now ensure that every quantifier is either limited or unlimited as follows. We proceed by induction on p . Suppose by induction that we have transformed the formula Φ into an equivalent formula Ψ with the property that Ψ is in formula-normal-form and all its quantifiers are either limited or unlimited. Now $\exists v_{p+1} \Psi$ is replaced by

$$[\exists^l v_{p+1} \Psi \vee \exists^u v_{p+1} \Psi].$$

Similarly, $\forall v_{p+1} \Psi$ is replaced by

$$[\forall^l v_{p+1} \Psi \wedge \forall^u v_{p+1} \Psi].$$

Now put the result into formula-normal-form. This completes the inductive step. Finally type-complete the quantifier free part. So the formula is now of the form

$$Q_k^{\alpha_k} v_k \cdots Q_1^{\alpha_1} v_1 \psi,$$

where ψ is $\bigvee_i \psi_i$, and each ψ_i is a type-completed conjunction of literals, and $\alpha_i \in \{l, u\}$. Note that it is equivalent to the original formula Φ .

Limited and unlimited literals: Suppose that ψ_i is a (not necessarily type-completed) conjunction of literals with the property that there is a *unique* (up to equivalence in $E(A)$) type-completed conjunction of literals ψ'_i equivalent in $E(\mathcal{A})$ to ψ_i . For example if ψ_i is $x \in A \wedge f(x) \notin A$ then ψ'_i is $x \in A \wedge f(x) \notin A \wedge IS_f(f(x)) \wedge_{g \neq f} \neg IS_g(f(x))$. In most cases, $\psi_i = \psi'_i$ will be type-completed itself.

Call a term t *limited (with respect to ψ_i)* if ψ'_i contains the literal $t \in A$, and *unlimited (with respect to ψ_i)* if ψ'_i contains the literal $t \notin A$. An equation or disequation is (un)limited in ψ_i if both sides of it are (un)limited in ψ_i . A tester predicate $IS_g(t)$ is (un)limited (with respect to ψ_i) if t is (un)limited (with respect to ψ_i). For the rest of the proof, when a term or (dis)equation is called limited, implicitly it is with respect to the type-completed ψ'_i in which it occurs. Recall a quantification Qv is called limited if it is of the form $(Qv \in A)$, also written $Q^l v$. If it is of the form $Qv \notin A$, also written $Q^u v$, it is unlimited.

The Algorithm: We are now ready to describe the algorithm. It takes as input a formula Φ of $E(\mathcal{A})$ of the form $\exists^u \bar{x} \chi(\bar{x}, \bar{y}, \bar{y}')$, where every quantification in χ is limited. So we may write Φ as $\exists^u \bar{x} Q_k^l x'_k \cdots Q_1^l x'_1 \psi(\bar{x}, \bar{x}', \bar{y}, \bar{y}')$. Also, recall our notation for variables: \bar{x} denotes existentially quantified unlimited variables and \bar{y} unlimited parameters; similarly, \bar{x}' denote quantified limited variables and let \bar{y}' denote limited parameters.

The algorithm proceeds in steps that transform the input formula to an equivalent output formula with additional syntactic properties. After describing each step we prove, unless obvious, termination and correctness of that step.

Step 1. Type Completion.

Input: An $E(\mathcal{A})$ -formula.

Output: An equivalent type completed formula.

So the formula is now of the form

$$\exists^u \bar{x} Q_k^l x'_k \cdots Q_1^l x'_1 \bigvee_i \psi_i(\bar{x}, \bar{x}', \bar{y}, \bar{y}'), \quad (*)$$

where each ψ_i is a type-completed conjunction of literals.

Step 2. Put Every Term into Term-Normal-Form.

Input: A type-completed formula in form $(*)$.

Output: An equivalent type-completed formula in form $(*)$ for which every term is in term-normal-form.

A term is in *term-normal-form* if it is of the form

$$t(v_1, \dots, v_{j+k})[v_1/L_1 w_1, \dots, v_j/L_j w_j]$$

where $t(\bar{v})$ is a term built from constructors only. Here L_i is a sequence of selectors applied to the variable w_i .

This step proceeds by pushing selectors past constructors as follows.

For each ψ_i do the following until no more apply. Pick some t occurring as a subterm in ψ_i . There are two cases.

Case 1: t Is Limited. Then for every constructor f , replace $f_j(t)$ in ψ_i by t , and

Case 2: t Is Unlimited. Let g be the unique constructor for which $\text{IS}_g(t)$ is a conjunct of ψ_i .

- For every $f \neq g$, replace $f_j(t)$ in ψ_i by t , and
- Say t is of the form $g(\bar{s})$ for some \bar{s} . Then replace $g_j(t)$ in ψ_i by s_j .

Termination: After applying each case the number of selectors in the formula decreases. Hence this step can be iterated only a finite number of times.

Correctness: By Lemma 1, the transformation preserves the equivalence of the formulas. Since only terms have changed, the resulting formula is still in formula-normal-form. Also, since every term in the resulting formula is already a term in the original formula, the result is also type-completed.

Step 3. Remove Selectors from All Unlimited Quantified Variables.

Input: A type-completed formula Φ in form (\star) for which every term is in term-normal-form.

Output: An equivalent type-completed formula for which every term is in term-normal-form and there are no selectors in front of unlimited quantified variables. That is one of the form

$$\exists^u \bar{x} Q_k^l x'_m \cdots Q_1^l x'_1 \bigvee_i \psi_i(\bar{x}, \bar{x}', \bar{y}, \bar{y}'), \quad (\dagger)$$

where each ψ_i is type-completed and no unlimited quantified variable x in ψ_i has a selector in front of it.

Recall \bar{x} is the collection of unlimited quantified variables. For each $x \in \bar{x}$ do the following until no more apply:

- Pick an $x \in \bar{x}$ for which Lx occurs in Φ where L is some non-empty block of selectors.
- Replace Φ by

$$\exists^u \bar{v} \bigvee_f \left[f(\bar{v}) \notin A \wedge (\exists^u \bar{x} Q_k^l x'_k \cdots Q_1^l x'_1 \bigvee_i \psi_i)[x/f(\bar{v})] \right],$$

where f varies over all constructors. Now remove the existential quantifier $\exists^u x$.

- Apply step 1 and then step 2.

Termination: The result of substituting $f(\bar{v})$ for x and then putting the terms in normal form results in every selector of the form Lx being transformed into one of the form $L'v_i$ where the length of L' is smaller than the length of L . And since steps 1 and 2 do not introduce selectors, the size of the largest block of selectors in front of unlimited quantified variables strictly decreases with each iteration.

Correctness: Once the procedure terminates no unlimited quantified variable x has a selector in front of it. That the output formula is equivalent follows from the fact that the following are equivalent in $E(\mathcal{A})$ for every formula Θ :

- $\exists^u x \Theta$.
- $\exists^u x \bigvee_f (\text{IS}_f(x) \wedge x \notin A \wedge \Theta)$.
- $\exists \bar{v} \exists^u x \bigvee_f (x = f(\bar{v}) \wedge \text{IS}_f(x) \wedge x \notin A \wedge \Theta)$.

Step 4a. Put Every (Dis)equation into Literal-Normal-Form.

Input: A formula of the form (\dagger) .

Output: An equivalent formula of the form (\dagger) in which every equation and disequation is in literal-normal-form.

An *unlimited (dis)equation is in literal-normal-form* if it is of the form

$$L_1 v_1 \Delta L_2 v_2$$

for some (possibly empty) L_i , and $\Delta \in \{=, \neq\}$. Here the v_i and $L_i v_i$ are unlimited.

A *limited (dis)equation is in literal-normal-form* if each term is in term-normal-form

$$t(v_1, \dots, v_{j+k})[v_1/L_1 y_1, \dots, v_j/L_j y_j],$$

with the additional property that the $v_{j+1}, \dots, v_{j+k}, L_1 y_1, \dots, L_j y_j$ are limited (that is, stated in ψ_i to be in A).

Throughout this step ensure that every literal $t \Delta s$ satisfies $t \in A$ if and only if $s \in A$ by applying the following steps whenever possible.

- An equation between terms t and s with $t \in A$ and $s \notin A$ is replaced with \perp .
- A disequation between terms t and s with $t \in A$ and $s \notin A$ is replaced with \top .

Hence every (dis)equation is either limited or unlimited.

For the unlimited (dis)equations repeat the following steps in each ψ_i until none can be applied; and then finally put the result into formula-normal-form, and type complete it.

- An unlimited equation of the form $f(\bar{t}) = g(\bar{s})$ is replaced with \perp if $f \neq g$ and with $\bigwedge_i t_i = s_i$ if $f = g$.
- An unlimited disequation of the form $f(\bar{t}) \neq g(\bar{s})$ is replaced with \top if $f \neq g$ and with $\bigvee_i t_i \neq s_i$ if $f = g$.
- An unlimited equation of the form $Ly = f(\bar{t})$, with y unquantified, is replaced with $\bigwedge_i f_i Ly = t_i$.

- An unlimited disequation of the form $Ly \neq f(\bar{t})$, with y unquantified, is replaced with $\bigvee_i f_i Ly \neq t_i$.

Termination: In general each item removes a literal $t\Delta s$ and replaces it with (a boolean combination of) a set of literals $\{t_i\Delta' s_i\}$. The relevant property here is that the largest number of constructors appearing in a term from $t\Delta s$ is strictly greater than the largest number of constructors appearing in a term from any of the $t_i\Delta' s_i$.

Correctness: The procedure produces a formula that is equivalent to the original one as seen from Lemma 1. The formula is of the form (\dagger) since the only introduced selectors are in front of unquantified unlimited variables from \bar{y} . Now if $t\Delta s$ is a resulting unlimited literal then it does not contain constructors. Also both t and s are in term-normal-form since each operation preserves being in term-normal-form. Hence $t\Delta s$ is in literal-normal-form.

Now we deal with the easier case of limited literals. Suppose t is limited (with respect to some ψ_i of the input formula). Note that t is already in term-normal-form. Now if some v_j or $L_j y_j$ occurring in t were unlimited (with respect to ψ_i) then t would also be unlimited, and so ψ_i is replaced with \perp . Hence every limited (dis)equation in the formula obtained is in literal-normal-form.

Step 4b. Put Tester Predicates into Literal-Normal-Form.

Input: A formula of the form (\dagger) in which every equation and disequation is in literal-normal-form.

Output: An equivalent formula in formula-normal-form, for which every literal is in literal-normal-form, and there are no selectors in front of unlimited quantified variables. Also it has the property that no literal mentions both a quantified unlimited variable from \bar{x} and a limited variable.

A tester predicate is in literal-normal-form if it is of the form $\text{IS}_g(Lv)$ for some g , where L is a possibly empty block of selectors, v is a variable and Lv is unlimited (that is, stated in ψ_i not to be in A).

We put every tester predicate into literal normal form by applying the following steps to every term t in ψ_i . Say t is of the form $f(\bar{s})$ for some \bar{s} and f .

- If t is limited then replace $\text{IS}_g(t)$ by \perp for every g .
- If t is unlimited then replace $\text{IS}_g(t)$ by \top if $f = g$ and by \perp otherwise.

Termination is clear in this case.

Correctness: By Lemma 1 the resulting formula is equivalent to the input formula. Every tester predicate is in normal form since each term in the input was in term-normal-form. Since in this step only tester predicates are removed the (dis)equations are still in literal-normal-form, and there are no selectors in front of unlimited quantified variables x from \bar{x} . We remark that although Φ is no longer necessarily type-completed, each disjunct ψ_i has a unique type-completion ψ'_i up to equivalence in $E(\mathcal{A})$. Recall that we call a term (un)limited in ψ_i if it is (un)limited in ψ'_i . So if x occurs in a term, that term is unlimited. Moreover if x occurs in an equation or disequation, then it is of the form $x\Delta Ly$, or $x\Delta x_1$, where x_1 is also from \bar{x} and y is unlimited and unquantified. Similarly

if x occurs in a tester predicate, it is of the form $\text{IS}_g(x)$ for some g . Hence no literal mentions both x and a limited variable.

Step 5. Separate the Unlimited Quantifiers from the Limited Quantifiers.

Input: A formula Φ of the form

$$\exists^u \bar{x} Q_k^l x'_k \cdots Q_1^l x'_1 \psi(x'_k, \dots, x'_1, \bar{x}, \bar{y}, \bar{y}'),$$

and with the properties resulting from the previous step.

Output: An equivalent formula in disjunctive normal form where each conjunction consists of formulae of the form

$$Q_k^l x'_k \cdots Q_1^l x'_1 \mu(x'_k, \dots, x'_1, \bar{y}, \bar{y}'), \quad \exists^u \bar{x} \delta(\bar{x}, \bar{y}) \text{ and } \epsilon(\bar{y}, \bar{y}').$$

Here

- μ is a possibly empty quantifier free formula with the property that every literal in μ is limited and mentions some variable from \bar{x}' .
- δ is a possibly empty conjunct of literals, and every literal in it is unlimited and mentions some variable from \bar{x} .
- ϵ is a possibly empty conjunct of literals.

In other words literals not in the scope of some quantifier are separated out. This can be done since by the previous step no literal mentions both some x and some limited variable x'_j or y' .

Step 6a. Remove Equations from δ That Mention x .

Input: The formula resulting from the previous step.

Output: An equivalent formula of the same form with the additional property that there are no equations in any of the δ .

Repeat the following in every δ until no more apply.

- Replace $x = x$ by \top and $x \neq x$ by \perp .
- If an equation $x = Ly$ is a conjunct in δ , then replace δ by $\delta[x/Ly]$.

Some literals may be transformed into literals of the form $L_1 y_1 \Delta L_2 y_2$. So put these new literals into the corresponding ϵ .

Termination: Since each stage removes the variable x from δ , this process eventually stops.

Correctness: After termination it is still the case that every literal in δ is unlimited and mentions some variable from \bar{x} . And the corresponding ϵ may have gained more literals. So the output formula has the same form but there are no equations left in δ since every equation mentioned some variable from \bar{x} .

Step 6b. Replace Each $\exists^u \bar{x} \delta$ with \top .

Input: The formula resulting from the previous substep.

Output: An equivalent formula without any unlimited quantification.

From the previous step there are no equations in any of the δ ; the disequations in δ are of the form $x \neq Ly$ or $x \neq x_i$ for some other unlimited quantified variable x_i from \bar{x} .

Let \bar{b} be an arbitrary instantiation of canonical terms for the parameters \bar{y} . We need to show that

$$E(\mathcal{A}) \models \exists^u \bar{x} \delta(\bar{x}, \bar{b})$$

First evaluate all selectors Lb by applying the definition of the selectors to the canonical terms from \bar{b} . Let $s \in \mathbb{N}$ be the maximum of the depth of the subterms (of the sentence) that do not mention variables from \bar{x} . For each f let $n_f \in \mathbb{N}$ be the number of distinct $x \in \bar{x}$ so that $IS_f(x)$ is a conjunct of δ . Choose $k \in \mathbb{N}$ larger than s and with the property that for every constructor $f \in \Sigma$ there are at least n_f distinct canonical terms of depth k that start with an f . This can be done since \mathcal{A} is not a total algebra.

Now let \bar{a} be distinct canonical terms of depth k that satisfy the type data in δ . Then $E(\mathcal{A}) \models \delta(\bar{a}, \bar{b})$. Indeed an unlimited literal $a_i \neq a_j$ holds in $E(\mathcal{A})$ by assumption that the elements of \bar{a} are distinct. An unlimited literal $a \neq b$ holds in $E(\mathcal{A})$ since the depth k of the term a is at least s which is greater than the depth of the term b . This completes the description and correctness of the algorithm.

Tracing through the proof, we see that we have transformed a formula of $E(\mathcal{A})$,

$$\exists^u \bar{x} Q_k^l x'_k \cdots Q_1^l x'_1 \bigvee \psi_i(\bar{x}, \bar{x}', \bar{y}, \bar{y}'),$$

into one of the form

$$\bigvee_j [\epsilon_j(\bar{y}, \bar{y}') \wedge Q_k^l x'_k \cdots Q_1^l x'_1 \mu_j(\bar{x}', \bar{y}', \bar{y})],$$

with only limited quantification, and where every literal in μ_j is limited and mentions some variable of \bar{x}' . This completes the proof of Lemma 2.

4.2 Corollaries and the Main Result

We can also give a characterisation of the definable relations of $E(\mathcal{A})$.

Theorem 2. *There is a procedure that given a formula $\Phi(\bar{y}, \bar{y}')$ of $E(\mathcal{A})$ returns an equivalent formula $\Phi'(\bar{y}, \bar{y}')$ with the property that every quantification is limited. In particular Φ' has the form*

$$\bigvee_j [\epsilon_j(\bar{y}, \bar{y}') \wedge Q_m^l x'_m \cdots Q_1^l x'_1 \mu_j(\bar{x}', \bar{y}', \bar{y})]$$

where ϵ_j is a type-completed conjunct of literals, μ_j is a type-completed quantifier free formula, and every literal in μ_j is limited and mentions some variable from $\bar{x}' = \cup_i \bar{x}'_i$.

Proof. Given a formula Φ of $E(\mathcal{A})$, first replace the formula with its type-completion. So it is now of the form

$$Q_m v_m \cdots Q_1 v_1 \psi,$$

where every quantifier is either limited or unlimited. Now pick an innermost formula of the form

$$Q^u \bar{x} Q_k^l x'_k \cdots Q_1^l x'_1 \psi(\bar{x}, \bar{x}', \bar{y}, \bar{y}'),$$

where the quantifiers Q_i^l are limited, ψ is in disjunctive normal form $\bigvee_i \psi_i$, where each ψ_i , a conjunction of literals, is type-completed. Note that k may be 0. Also, we may assume that $Q^u \bar{x}$ is $\exists^u \bar{x}$, for if it were $\forall^u \bar{x}$ then replace it with $\neg \exists^u \bar{x} \neg$, and push the second \neg inward as usual. Applying the lemma results in a formula with no unlimited quantification. Now repeat this process until there are no more unlimited quantifiers in Φ . Finally although the lemma may result in a formula containing a conjunct of literals B that is not type-completed, it is the case that B is equivalent in $E(\mathcal{A})$ to a type-completed conjunction of literals B' . So replace B by B' . This completes the proof.

A formula $\Phi(\bar{y}, \bar{y}')$ of $E(\mathcal{A})$ is called an *\mathcal{A} -formula* if it is type-completed, and of the form

$$Q_k^l x'_k \cdots Q_1^l x'_1 \mu(\bar{x}', \bar{y}', \bar{y}),$$

where every literal in μ is limited. Recall this means that each term is limited and of the form

$$t(v_1, \dots, v_{j+k})[v_1/L_1 y_1, \dots, v_j/L_j y_j],$$

where each of $v_{j+1}, \dots, v_{j+k}, L_1 y_1, \dots, L_j y_j$ is limited, and $t(\bar{v})$ consists of constructors only.

Observe that if χ_1 and χ_2 are \mathcal{A} -formulae, then they are equivalent in \mathcal{A} if and only if they are equivalent in $E(\mathcal{A})$. Also the subformulae from Theorem 2 of the form

$$Q_k^l x'_k \cdots Q_1^l x'_1 \mu(\bar{x}', \bar{y}', \bar{y}),$$

are \mathcal{A} -formulae. Hence we have the next corollary.

Corollary 1. *If \mathcal{A} admits elimination of quantifiers, then so does $E(\mathcal{A})$.*

We now restate, and are ready to prove the main theorem.

Theorem 3. *If a partial algebra \mathcal{A} has decidable QEP then so does its free total extension $E(\mathcal{A})$.*

Proof. Given a formula $\Phi(\bar{v})$ and a tuple of elements \bar{w} from $E(\mathcal{A})$. Apply Theorem 2 and transform Φ into Φ' . Now form the sentence $\Phi'(\bar{w})$. This sentence consists of quantifier free sentences $\epsilon(\bar{a}, \bar{a}')$, and sentences of the form

$$Q_k^l v'_k \cdots Q_1^l v'_1 \bigvee_i \psi_i(\bar{v}', \bar{a}', \bar{a}),$$

where each ψ_i consists of limited literals and type data. Here \bar{a}' and \bar{a} are amongst \bar{w} , and moreover $\bar{a}' \subset A$ and $\bar{a} \cap A = \emptyset$. Also $\bar{v}' = \cup_j v'_j$.

Now evaluate $\epsilon(\bar{a}, \bar{a}')$. This is done by first applying the definition of the selector functions and tester predicate and then applying the fact that $t_1 = t_2$,

where the t_i are canonical terms not in A , if and only if t_1 and t_2 are syntactically equal.

This leaves an \mathcal{A} -sentence that is evaluated using the algorithm for the theory of \mathcal{A} .

5 Application

Recall that $GT_\Sigma(C)$ denotes the (total) algebra of ground terms generated by the non-empty set C of constants from Σ . We start with the following definition.

Definition 2. *Let E be a set of ground equations; that is equations of the form $t = s$ where t and s are ground terms. Consider the quotient of GT_Σ by the smallest congruence generated by E . This is a (total) algebra (over Σ) that we will denote by \mathcal{A}_E . Call a total algebra **finitely presented** if it is of the form \mathcal{A}_E for some finite set E of ground equations.*

Easy examples include GT_Σ itself and every finite algebra. Decision problems for finitely presented algebras in a given variety of algebras have received much attention. For instance, the word problem in finitely presented semigroups or groups (in the variety of semigroups or groups) is, in general, undecidable. However in the variety of *all* algebras, the situation is different. For instance Kozen in [10] considers the uniform word problem, the finiteness problem, the subalgebra membership problem and the triviality problem: all are decidable in polynomial time. Comon in [3] proves that the first order theory of any finitely presented term algebra is decidable. The proof uses algebraic techniques combined with quantifier elimination methods. Our main theorem can now be applied to give another and, we think, simpler proof to decide the first order theory for finitely presented term algebras.

The relationship between finitely presented algebras and free total extensions is described in the next theorem (implicit in [11]).

Theorem 4. *A total algebra is a finitely presented term algebra if and only if it is the free total extension of a finite partial algebra.*

So we immediately have the following application of Theorem 1.

Theorem 5. *Let \mathcal{A}_E be a finitely presented algebra. Then it has decidable QEP. In particular its first order theory is decidable.*

Acknowledgement

We thank the referees for their helpful comments used in improving the presentation and proof of the results.

References

1. E. Astesiano, M. Bidoit, H. Kirchner, B. Krieg-Brückner, P.D. Mosses, D. Sannella, A. Tarlecki: CASL: The Common Algebraic Specification Language. *Theoretical Computer Science* 286:2 (2002) 153–196
2. A. Blumensath, E. Gradel: Automatic structures. *Proceedings of the 15th Annual IEEE Symposium on Logic in Computer Science (LICS) 2000* 51–62
3. H. Comon: Complete axiomatization of some quotient term algebras: *Theoretical Computer Science* 122:1-2 (1993) 165–200
4. K. Compton, C. Henson: A uniform method for proving lower bounds on computational complexity of logical theories. *Annals of Pure and Applied Logic* 48 (1990) 1–79
5. W. Hodges: *Model theory*. Cambridge University press (1993)
6. B. Khoussainov, S. Rubin, F. Stephan: Automatic Linear Orders and Trees. *Transactions on Computational Logic (TOCL) special issue (selected papers from the LICS 2003 conference)*, editor Phokion G. Kolaitis (accepted)
7. B. Khoussainov, A. Nerode: Automatic Presentations of Structures. D. Lieviant (editor) *Proceedings of the conference on Logic and Computational Complexity* (1994) vol. 960 of LNCS (1995) 367–393
8. K. Korovin, A. Voronkov: A decision procedure for the existential theory of term algebra with the Knuth-Bendix ordering. In *Proceedings IEEE Conference on Logic in Computer Science* (2000) 291–302
9. G. Grätzer: *Universal Algebra*. D. Van Nostrand Co., Inc., Princeton, N.J.-Toronto, Ont.-London (1968)
10. D. Kozen: Complexity of finitely presented algebras. In *Proceedings of the 9th ACM symposium on theory of computing* (1977) 164–177
11. D. Kozen: Partial automata and finitely generated congruences: an extension of Nerode's theorem. In J. Crossley, J. Remmel, R. Shore, M. Sweedler editors. *Logical methods: in honour of Anil Nerode's Sixtieth Birthday*, Birkhauser (1993) 490–511
12. V. Kuncak, M. C. Rinard: Structural subtyping of non-recursive types is decidable. In *Proceedings IEEE Conference on Logic in Computer Science* (2003) 96–107
13. M. Lohrey: Automatic structures of bounded degree. In *proceedings of 10th International conference Logic for programming, artificial intelligence and reasoning*, editors M. Vardi and A. Voronkov, vol. 2850 of LNCS (2003) 346–360
14. M. Maher: A CLP view of logic programming. In *Algebraic and Logic Programming* (1992) 364–383
15. A. Mal'cev: Axiomatizable classes of locally free algebras of various types. In *The Metamathematics of Algebraic Systems. Anatolii Ivanovič Mal'cev. Collected papers: 1936-1967*, B. Wells III, Ed. Vol. 66. North Holland. Chapter 23 (1971) 262–281
16. T. Zhang, H. Sipma, Z. Manna: Term algebras with length function and bounded quantifier alternation. *the 17th International Conference on Theorem Proving in Higher Order Logics (TPHOLs'04)* Volume 3223 of LNCS (2004) 321–336
17. T. Rybina, A. Voronkov: *A Decision procedure for term algebras with queues*. ACM Transaction on Computational Logic 2:2 (2001) 155–181
18. S. Vorobyov: An improved lower bound for the elementary theories of trees. In *Proceedings of the 13th Intl. Conference on automated deduction*, Volume 1104 of LNCS (1996) 275–287
19. I. Walukiewicz: Monadic second order logic on tree-like structures. *Theoretical computer science* 275:1-2 (2002) 311–346

Automatic Linear Orders and Trees

BAKHADYR KHOUSSAINOV and SASHA RUBIN

University of Auckland

and

FRANK STEPHAN

National University of Singapore

We investigate partial orders that are computable, in a precise sense, by finite automata. Our emphasis is on trees and linear orders. We study the relationship between automatic linear orders and trees in terms of rank functions that are related to Cantor–Bendixson rank. We prove that automatic linear orders and automatic trees have finite rank. As an application we provide a procedure for deciding the isomorphism problem for automatic ordinals. We also investigate the complexity and definability of infinite paths in automatic trees. In particular, we show that every infinite path in an automatic tree with countably many infinite paths is a regular language.

Categories and Subject Descriptors: F.1.1 [**Computation by Abstract Devices**]: Models of Computation—*Automata* (e.g., *finite, push-down, resource-bounded*); F.4.3 [**Mathematical Logic and Formal Languages**]: Formal Languages—*Classes defined by grammars or automata* (e.g., *context-free languages, regular sets, recursive sets*)

General Terms: Theory, Languages

Additional Key Words and Phrases: Automatic structures, trees, linear orders

1. INTRODUCTION

Consider a class of infinite structures, such as the class of graphs, partial orders, trees, groups, or lattices, etc. A given structure in this class may or may not be computable. If it is, one then naturally asks whether or not the structure,

B. Khoussainov was partially supported by a Marsden fund of the New Zealand Royal Society. Most work was done while F. Stephan held previous positions at the University of Heidelberg and the National ICT Australia. He was supported by the Deutsche Forschungsgemeinschaft (DFG), Heisenberg grant Ste 967/1-1.

National ICT Australia is funded by the Australian Government's Department of Communications, Information Technology and the Arts and the Australian Research Council through Backing Australia's Ability and the ICT Centre of Excellence Program.

Author's addresses: B. Khoussainov, S. Rubin, Department of Computer Science, The University of Auckland, Private Bag 92019, Auckland, New Zealand; email: {bmk,rubin}@cs.auckland.ac.nz; F. Stephan, School of Computing and Department of Mathematics, National University of Singapore, 3 Science Drive 2, Singapore 117543; email: fstephan@comp.nus.edu.sg.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or direct commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other works requires prior specific permission and/or a fee. Permissions may be requested from Publications Dept., ACM, Inc., 1515 Broadway, New York, NY 10036 USA, fax: +1 (212) 869-0481, or permissions@acm.org.

© 2005 ACM 1529-3785/05/1000-0675 \$5.00

or algorithmic problems of the structure, are feasibly computable. In case that ‘feasible structure’ means computable by finite automata, one has an *automatic structure* (Definition 2.4). The automata in this article operate synchronously on finite words. Using the closure of these automata under Boolean operations and projection, one has that the first-order theory of an automatic structure is decidable, see, for instance, Khoussainov and Nerode [1994]. From a computer science point of view, this result suggests that automatic structures may be suitable objects that can be effectively queried. This is illustrated in the related concept of automatic groups from computational group theory [Epstein et al. 1992]. There it is proven that a finitely generated automatic group is finitely presentable and that its word problem is solvable in quadratic time. The general notion of structures presentable by automata has been recently studied in Blumensath [1999], Blumensath and Grädel [2000], Delhommé [2004], Delhommé et al. [2002], Ishihara et al. [2002], Khoussainov and Nerode [1994], Khoussainov and Rubin [2001, 2003], Khoussainov et al. [2004a, 2004b], Kuske [2003] and Lohrey [2003]. Throughout this article, we will use the following more general theorem proved in Blumensath and Grädel [2000] without explicit mention.

THEOREM 1.1. *Given an automatic structure \mathcal{A} and a relation R , which is first-order definable in \mathcal{A} (with the quantifier \exists^∞ which stands for ‘there exist infinitely many’), one can effectively construct an automaton recognizing R .*

Our work is motivated by the following general problem:

Problem 1.2. Given a class of structures \mathcal{C} , characterize the isomorphism types of the automatic structures in \mathcal{C} .

The isomorphism type of a structure \mathcal{A} is defined as the set of structures that are isomorphic to \mathcal{A} . A satisfactory answer to this problem would give a non-automata-theoretic description of those isomorphism types that contain an automatic structure. We note that the isomorphism problem for automatic graphs is Σ_1^1 -complete [Khoussainov et al. 2004a] and that the isomorphism problem for the class of all automatic structures has the same complexity. Consequently, we restrict the class of structures under consideration for better results; for instance, we show that the isomorphism problem of automatic ordinals (represented as automatic well-orderings) is recursive. This article is concerned with the class of partially ordered structures, with an emphasis on trees and linear orderings. A *partial order* (*partial ordering*) is a structure (A, \leq) such that \leq is a reflexive, transitive and anti-symmetric binary relation on the domain A . A *linear order* \mathcal{L} is a partial order (L, \leq) in which \leq is total, that is $(\forall x)(\forall y)[x \leq y \vee y \leq x]$.

Classically linear orderings are characterized in terms of scattered and dense linear orderings as follows: One says that \mathcal{L} is *dense* if for all distinct a and b in L with $a < b$ there exists an $x \in L$ with $a < x < b$. There are only five types of countable dense linear orderings up to isomorphism: the order of rational numbers with or without least or greatest elements, and the order type of the trivial linear order with exactly one element. One says that \mathcal{L} is *scattered* if it does not contain a nontrivial dense subordering. Examples of scattered

linear orders are finite sums (see Definition 3.1) of Cartesian products of ω (the order type of the natural numbers) and ζ (the order type of the integers). The following theorem is the classical representation of countable linear orderings—it is proved below.

THEOREM 3.2. *Every countable linear ordering \mathcal{L} can be represented as a dense sum of countable scattered linear orderings.*

The scattered linear orderings can be characterized inductively whereby to each scattered linear order \mathcal{L} one associates a countable ordinal—called the VD–rank of \mathcal{L} (Definition 3.3), a version of Cantor–Bendixson rank for topological spaces. One of the results in this paper (Proposition 4.6) says that the VD–rank of every automatic scattered linear order is finite.

Definition 3.8 introduces the FC–rank of a (not necessarily scattered) linear order, with the property that FC–rank and VD–rank agree on scattered linear orders. Combining Proposition 4.6 and Theorem 3.2 gives the following necessary condition for automatic linear orders.

THEOREM 4.7. *If \mathcal{L} is an automatic linear order, then its FC–rank is finite.*

This condition is not sufficient for a linear order to be automatic; indeed, there is a scattered linear order of FC–rank 2 that is not isomorphic to any automatic linear order (Remark 4.8).

The proof of Theorem 4.7 generalizes a novel technique of Delhommé. In a manuscript circulated in 2001, he gives a full characterization of automatic ordinals.

COROLLARY [DELHOMMÉ 2004]. *An ordinal α is isomorphic to an automatic ordinal if and only if $\alpha < \omega^\omega$.*

There is an effective procedure that given an automatic presentation of an ordinal, produces the ordinal’s Cantor–normal–form.

THEOREM 5.3. *The isomorphism problem for automatic ordinals is decidable.*

Recently, Delhommé [2004] independently generalized his technique and notes that Theorem 4.7 can be obtained as a corollary. He also generalizes the technique to tree-automatic structures and in particular characterizes the tree-automatic ordinals.

The second topic of this paper concerns the class of partial orders known as trees. A *tree* $T = (T, \preceq)$ is a partial order that has a minimum element and in which every set of the form $\{y \in T \mid y \preceq x\}$ forms a finite linear order. Elements of trees are called nodes. A *path* of a tree (T, \preceq) is a subset $P \subseteq T$, which is linearly ordered and maximal (with respect to set theoretic inclusion) with this property. An *infinite path* is a path P consisting of infinitely many nodes. We are interested in understanding algebraic, model-theoretic as well as computational properties of automatic trees.

We deal with trees by associating to each tree its Kleene–Brouwer ordering. This transformation preserves automaticity, and associates the Cantor–Bendixson rank (CB–rank for short) of trees with the FC–ranks of the associated

linear orders. Informally the CB-rank of the tree tells us how complicated its infinite paths are in terms of ordinals (see e.g., Kechris [1995]). This relationship between trees and linear orders yields the next result.

THEOREM 7.10. *The CB-rank of an automatic tree is finite.*

It is known that every infinite finitely branching tree has an infinite path—usually referred to as König’s Lemma. The proof of this fact does not produce an infinite path constructively. In fact, there are even examples of computable finitely branching trees with *exactly* one infinite path, and that path is *not* computable. Moreover, if one omits the assumption that the tree is finitely branching, then there are examples of computable trees in which every infinite path is not even arithmetical (see Rogers [1967]). This negative phenomenon fails dramatically when one considers automatic trees, and not only finitely branching ones. To start with, here is an automatic version of König’s Lemma.

THEOREM 8.2. *Every infinite automatic finitely branching tree (T, \preceq) has a regular infinite path. That is, there exists a regular set $P \subseteq T$ so that P is an infinite path of the tree.*

This is because the length-lexicographically leftmost path is definable using the quantifier \exists^∞ in (T, \preceq) . We can significantly strengthen this theorem under the assumption that the tree has at most countably many paths. Indeed from Theorem 7.10, we derive that if an automatic finitely branching tree T has countably many infinite paths then every path of T is regular (Theorem 8.3). This is because the set of paths in such trees is definable. Moreover, one may even omit the assumption that the tree be finitely branching.

THEOREM 8.7. *Every infinite path in an automatic tree with countably many infinite paths is regular.*

2. PRELIMINARIES

All classical definitions and results on linear orderings can be found in Rosenstein [1982]. Countable means finite or countably infinite. All structures are assumed to be countable. Definable means first-order definable with the additional quantifier \exists^∞ .

A thorough introduction to automatic structures can be found in Blumensath [1999] and Khoussainov and Nerode [1994]. A survey paper of Khoussainov and Rubin [2003] discusses the basic results and possible directions for future work in the area. Familiarity with the basics of finite automata theory is assumed though for completeness and to fix notation the necessary definitions are included here.

A *finite automaton* \mathcal{A} over an alphabet Σ is a tuple (S, ι, Δ, F) , where S is a finite set of *states*, $\iota \in S$ is the *initial state*, $\Delta \subseteq S \times \Sigma \times S$ is the *transition table* and $F \subseteq S$ is the set of *final states*. A *computation* of \mathcal{A} on a word $\sigma_1\sigma_2 \dots \sigma_n$ ($\sigma_i \in \Sigma$) is a sequence of states, say q_0, q_1, \dots, q_n , such that $q_0 = \iota$ and $(q_i, \sigma_{i+1}, q_{i+1}) \in \Delta$ for all $i \in \{0, 1, \dots, n - 1\}$. If $q_n \in F$, then the computation is *successful*. If a word has a successful computation, then we say that automaton \mathcal{A} *accepts* the word. The *language* accepted by the automaton \mathcal{A} is the set of all words

accepted by \mathcal{A} . In general, $D \subseteq \Sigma^*$ is *finite automaton recognizable*, or *regular*, if D is equal to the language accepted by \mathcal{A} for some finite automaton \mathcal{A} . An automaton \mathcal{A} is *deterministic* if, for every $q \in S$ and $\sigma \in \Sigma$, there is a unique $q' \in S$ such that $(q, \sigma, q') \in \Delta$.

We briefly mention Büchi automata as they will be used later in Lemma 8.6. A (nondeterministic) Büchi automaton (S, ι, Δ, F) over Σ accepts an infinite string $\alpha \in \Sigma^\omega$ if it has a run $(q_i)_{i \in \mathbb{N}}$ such that there is some state $f \in F$ with $f = q_j$ for infinitely many $j \in \mathbb{N}$. The language accepted by such an automaton is called *Büchi recognizable*. See, for instance, Khoussainov and Nerode [2001] for a modern treatment.

Classically finite automata recognize sets of words. The following definition extends recognizability to relations of arity n , by *synchronous n -tape automata*. Informally, a synchronous n -tape automaton can be thought of as a one-way Turing machine with n input tapes [Eilenberg et al. 1969]. Each tape is regarded as semi-infinite, having written on it a word in the alphabet Σ followed by an infinite succession of blanks, \diamond symbols. The automaton starts in the initial state, reads simultaneously the first symbol of each tape, changes state, reads simultaneously the second symbol of each tape, changes state, etc., until it reads a blank on each tape. The automaton then stops and accepts the n -tuple of words if it is in a final state. The set of all n -tuples accepted by the automaton is the relation recognized by the automaton. Here is a formalization:

Definition 2.1. Write Σ_\diamond for $\Sigma \cup \{\diamond\}$ where \diamond is a symbol not in Σ . The *convolution* of a tuple $(w_1, \dots, w_n) \in \Sigma^{*n}$ is the string $\otimes(w_1, \dots, w_n)$ of length $\max_i |w_i|$ over alphabet $(\Sigma_\diamond)^n$ defined as follows. Its k th symbol is $(\sigma_1, \dots, \sigma_n)$ where σ_i is the k th symbol of w_i if $k \leq |w_i|$ and \diamond otherwise.

The convolution of a relation $R \subseteq (\Sigma^*)^n$ is the relation $\otimes R \subseteq ((\Sigma_\diamond)^n)^*$ formed as the set of convolutions of all the tuples in R .

Definition 2.2. An n -tape automaton on Σ is a finite automaton over the alphabet $(\Sigma_\diamond)^n$. An n -ary relation $R \subseteq \Sigma^{*n}$ is *finite automaton recognizable* or *regular* if its convolution $\otimes R$ is recognizable by an n -tape automaton.

For instance, let \preceq_p be the *prefix* relation. That is, for $x, y \in \Sigma^*$ define $x \preceq_p y$ if there exists $z \in \Sigma^*$ such that $xz = y$. If z is not the empty string ϵ , then x is a *proper prefix* of y , written $x \prec_p y$. So \preceq_p and \prec_p are regular binary relations over Σ . For example, if $\Sigma = \{0, 1\}$, then $\otimes(\preceq_p) = \{\begin{pmatrix} 1 \\ 1 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \end{pmatrix}\}^* \{\begin{pmatrix} 0 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \end{pmatrix}\}^*$.

PROPOSITION 2.3 (KHOUSSAINOV AND NERODE [1994]). *n -tape automata can be effectively determinised, and are effectively closed under Boolean operations and projection.*

We now relate n -tape automata to structures. A *structure* \mathcal{A} consists of a set A called the *domain* and some constants, relations and operations on A . We may assume that \mathcal{A} only contains relational predicates as the operations can be replaced with their graphs and constants can be thought of as operations of arity 0. We write $\mathcal{A} = (A, R_1^A, \dots, R_k^A)$ where R_i^A is an n_i -ary relation on A . The *signature* of \mathcal{A} is (R_1, \dots, R_k) . A structure is finite (countably infinite) if its domain has a finite (countably infinite) number of elements. An *isomorphism*

between structures \mathcal{A} and \mathcal{B} of the same signature is a bijective mapping $\nu : A \rightarrow B$ such that for every relational symbol from the signature, say R of arity i , $(a_1, \dots, a_i) \in R^A$ if and only if $(\nu(a_1), \dots, \nu(a_i)) \in R^B$ for every tuple (a_1, \dots, a_i) .

Definition 2.4. A structure \mathcal{A} is *automatic over Σ* if its domain $A \subseteq \Sigma^*$ and the relations $R_i^A \subseteq (\Sigma^*)^{n_i}$ are finite automaton recognizable.

An isomorphism from a structure \mathcal{B} to a structure \mathcal{A} that is automatic over Σ is an *automatic presentation* of \mathcal{B} in which case \mathcal{B} is called *automatically presentable* over Σ . A structure is called *automatic (automatically presentable)* if it is automatic (automatically presentable) over some alphabet.

For instance, the structure (Σ^*, \leq_p) is automatic. Other examples of automatically presentable structures are Presburger arithmetic $(\mathbb{N}, S, +, 0)$, the group of integers $(\mathbb{Z}, +)$, and the Boolean algebra of finite or co-finite subsets of \mathbb{N} .

We now mention some important examples of automatic linear orders. Fix an ordering on Σ , say $\sigma_1 < \sigma_2 < \dots < \sigma_k$. Define x *lexicographically less* than y , written $x <_{lex} y$, if either x is a proper prefix of y , or else in the first place where they differ the symbol in x is $<$ the symbol in y . Then, (Σ^*, \leq_{lex}) is an automatic linear order. Also define x *length-lexicographically less* than y , written $x <_{llex} y$, if $|x| < |y|$ or else $|x| = |y|$ and $x <_{lex} y$. Then (Σ^*, \leq_{llex}) is an automatic linear order of type ω .

If $\mathcal{A} = (A, (R_i)_i)$ is an automatic structure over Σ , then $(A, (R_i)_i, \leq_{lex}, \leq_{llex})$ is also automatic over Σ . Consequently, every automatic presentation of \mathcal{A} can be expanded to include the regular relations \leq_{lex} and \leq_{llex} (restricted to the domain A). This fact will be used repeatedly.

Examples of automatically presentable linear orders are (\mathbb{N}, \leq) , (\mathbb{Z}, \leq) and the order on the rationals (\mathbb{Q}, \leq) . Moreover, if $\mathcal{L}_1 = (L_1, \leq_1)$ and $\mathcal{L}_2 = (L_2, \leq_2)$ are automatic linear orders, then so is their sum and their product. Hence (the ordering given by) the ordinal ω^n is automatically presentable for every $n \in \mathbb{N}$. An example of such a presentation is the lexicographical ordering \leq_{lex} restricted to the domain $(1^*0)^n$.

Below, we present two generic examples of automatic trees.

Example 2.5. Let R be a nonempty regular language and let $Pref(R)$ be the set of prefixes of strings in R . Recall the prefix relation \leq_p . Then, the partial orders $(Pref(R), \leq_p)$ and $(R \cup \{\epsilon\}, \leq_p)$ are automatic trees.

Example 2.6. Let R be a regular language containing ϵ . Consider the partial order $\mathcal{T} = (R, \leq)$, where $x \leq y$ iff $x = y$ or $|x| < |y|$ and x is lexicographically smallest among all $x' \in R$ such that $|x'| = |y|$. Note that by Proposition 2.3, the relation \leq is regular since it is first-order definable from regular predicates. Hence, \mathcal{T} is an automatic tree.

3. LINEAR ORDER PRELIMINARIES

If \mathcal{L} is a linear ordering, then, unless specified, we denote its domain by L and ordering by \leq_L or simply \leq . Write $<$ for the corresponding strict order; that is, $x < y$ is defined by $x \leq y \wedge x \neq y$. If $S \subseteq L$, then we write $\mathcal{S} = (S, \leq_S)$ for the ordering with domain S and ordering \leq restricted to S . In this case, we say that \mathcal{S} is a *subordering* of \mathcal{L} .

Write ω for the (order) type of the positive integers, ω^* for the negative integers, ζ for the integers, η for the rationals and \mathbf{n} for the finite order on n elements. The empty ordering is written $\mathbf{0}$ and the ordering with exactly one element is written $\mathbf{1}$. A subordering S of \mathcal{L} is an *interval* if for every $x, y \in S$ with $x <_L y$ it is the case that $z \in S$ for every $z \in L$ satisfying $x <_L z <_L y$. An interval is *closed* if it is of the form $\{z \in L \mid x \leq z \leq y\}$ if $x \leq y$ and $\{z \in L \mid y \leq z \leq x\}$ otherwise; either way the interval is written $[x, y]$.

Definition 3.1. Consider a linear order \mathcal{I} as an index set for a set of linear orderings $\{\mathcal{A}_i\}_{i \in I}$. The \mathcal{I} -sum

$$\mathcal{L} = \Sigma\{\mathcal{A}_i \mid i \in I\}$$

is the linear order with domain $\cup_i A_i$ (we may assume that the domains A_i are pairwise disjoint). For $x \in A_i, y \in A_j$, define $x \leq_L y$ if $(i <_I j) \vee (i = j \wedge x \leq_{A_i} y)$.

We refer to the case when I is dense as a *dense sum*. Classically linear orderings are characterized in terms of scattered and dense linear orderings. Recall that \mathcal{L} is *scattered* if it does not contain a nontrivial dense subordering. If every \mathcal{A}_i is scattered, and \mathcal{I} is scattered, then the sum is scattered. If $\mathcal{A}_i = \mathcal{B}$ for every $i \in I$, then the sum is written as a product $\mathcal{B}\mathcal{I}$. For instance, $\omega 2$ is $\omega + \omega$. The following classical characterization, whose proof is given below, is due to Hausdorff [1908].

Theorem 3.2. Every countable linear ordering \mathcal{L} can be represented as a dense sum of countable scattered linear orderings.

In turn, the scattered linear orders can be characterized inductively, where to each linear order one associates an ordinal ranking, called the VD-rank. VD stands for very discrete.

Definition 3.3. For each countable ordinal α , define the set VD_α of linear orders inductively as

- (1) $\text{VD}_0 := \{\mathbf{0}, \mathbf{1}\}$.
- (2) $\text{VD}_\alpha :=$ all linear orderings formed as \mathcal{I} -sums where \mathcal{I} is of the type ω, ω^*, ζ or \mathbf{n} for some $n < \omega$ and every \mathcal{A}_i is a linear ordering from $\bigcup\{\text{VD}_\beta \mid \beta < \alpha\}$.

Define the class VD as the union of the VD_α . The *VD-rank* of a linear ordering $\mathcal{L} \in \text{VD}$, written $\text{VD}(\mathcal{L})$, is the least ordinal α such that $\mathcal{L} \in \text{VD}_\alpha$.

So the only linear orders of VD-rank 0 are $\mathbf{0}$ and $\mathbf{1}$. The linear orders of VD-rank 1 are exactly those of type ω, ω^*, ζ and \mathbf{n} for $1 < n < \omega$.

Example 3.4. Let $\mathcal{L}_1 = \Sigma\{\zeta + \mathbf{n} \mid n \in \omega\}$, $\mathcal{L}_2 = (\zeta \cdot \zeta) \cdot \zeta$. Then $\text{VD}(\mathcal{L}_1) \leq 2$ since \mathcal{L}_1 can be re-expressed as an ω -sum of orders of VD-rank 1. And in fact $\text{VD}(\mathcal{L}_1) = 2$ since \mathcal{L}_1 does not have rank 1. Similarly, $\text{VD}(\mathcal{L}_2) = 3$ and $\text{VD}(\mathcal{L}_1 + \mathcal{L}_2) = 4$.

More generally, if $\alpha = \max(\text{VD}(\mathcal{L}_1), \text{VD}(\mathcal{L}_2))$, then $\alpha \leq \text{VD}(\mathcal{L}_1 + \mathcal{L}_2) \leq \alpha + 1$ (see Rosenstein [1982, Lemma 5.15]).

Example 3.5. Let α, β be countable ordinals. Then $\text{VD}(\beta) \leq \alpha$ iff $\beta \leq \omega^\alpha$. In particular, $\text{VD}(\omega^\alpha) = \alpha$.

THEOREM 3.6 [HAUSDORFF 1908]. A countable linear ordering \mathcal{L} is scattered if and only if \mathcal{L} is in VD.

There is an alternative definition of ranking that generalizes VD-rank and assigns an ordinal rank to nonscattered linear orders as well. We proceed with the definitions.

Definition 3.7. A condensation (map) of \mathcal{L} is a mapping c from L to nonempty intervals of L such that $c(y) = c(x)$ whenever $y \in c(x)$. The condensation of \mathcal{L} is the linear order $c[\mathcal{L}]$ whose domain consists of the collection of nonempty intervals $c(x)$ for $x \in L$ ordered by $c(x) \trianglelefteq c(y)$ if $c(x) = c(y)$ or $(\forall x' \in c(x))(\forall y' \in c(y))[x' <_L y']$.

The relation x is condensed (by c) to y defined as $x \in c(y)$ is an equivalence relation.

As an illustration of the definition, we prove that every countable linear ordering can be represented as a dense sum of scattered linear orderings (Theorem 3.2).

PROOF. The mapping $c_S : x \mapsto \{y \in L \mid [x, y] \text{ is scattered}\}$ is a condensation since if $y \in c_S(x)$ then for all $a, [y, a]$ does not contain a dense subordering if and only if $[x, a]$ does not contain a dense subordering. Now $\mathcal{L} = \sum \{a \mid a \in c[\mathcal{L}]\}$ and each $a = c_S(x) \in c[\mathcal{L}]$ is scattered. Finally, $c_S[\mathcal{L}]$ is dense since for $c_S(x) \triangleleft c_S(y)$, if there is no z with $c_S(x) \triangleleft c_S(z) \triangleleft c_S(y)$ then $[x, y]$ is scattered. \square

Definition 3.8. Define $c_{FC}(x)$ as $\{y \in L \mid [x, y] \text{ is a finite interval of } \mathcal{L}\}$. For every ordinal α , define a condensation map c_{FC}^α of \mathcal{L} inductively:

$$c_{FC}^\alpha(x) = \left\{ y \in L : y = x \vee (\exists \beta < \alpha) [c_{FC}(c_{FC}^\beta(y)) = c_{FC}(c_{FC}^\beta(x))] \right\}.$$

In the expression $c_{FC}(c_{FC}^\beta(y))$, the term c_{FC} is a condensation map of the linear order $c^\beta[\mathcal{L}]$; hence, $c_{FC}(c_{FC}^\beta(y))$ is the set of elements of $c^\beta[\mathcal{L}]$ that are condensed to the element $c_{FC}^\beta(y)$. Note that this definition implicitly gives $c_{FC}^0(x) = \{x\}$ and $c_{FC}^1(x) = c_{FC}(x)$.

Here FC stands for finite condensation and indeed c_{FC}^α is a condensation map of \mathcal{L} . The idea is that $c_{FC}^1(x)$ is the set of elements of \mathcal{L} that are only finitely far away from x ; $c_{FC}^2(x)$ is the set of elements of \mathcal{L} that are in intervals of $c_{FC}[\mathcal{L}]$ which themselves are only finitely far away in $c_{FC}[\mathcal{L}]$ from the interval $c_{FC}^1(x)$, etc.

Definition 3.9. The least ordinal α such that $c_{FC}^\beta(x) = c_{FC}^\alpha(x)$ for all $x \in L$ and $\beta \geq \alpha$ is called the FC-rank of \mathcal{L} , written $\text{FC}(\mathcal{L})$.

For instance, a nonempty linear order \mathcal{L} is dense if and only if its FC-rank is 0. A dense sum of orders of FC-rank α has FC-rank α . **From now on, we write c for c_{FC} .**

Example 3.10. The FC-rank of \mathcal{L} is the least ordinal α such that $c^\alpha[\mathcal{L}]$ is dense. So \mathcal{L} is scattered if and only if $c^\alpha[\mathcal{L}] \simeq \mathbf{1}$ for some ordinal α .

The following theorem connects FC-ranks and VD-ranks of scattered linear orderings.

THEOREM 3.11 [HAUSDORFF 1908]. *If \mathcal{L} is scattered, then its VD-rank equals its FC-rank.*

For instance, the ordinal ω^n is scattered and has VD-rank and FC-rank n .

Given linear order \mathcal{L} and $A \subseteq L$, we will use c to denote the condensation of \mathcal{L} and c_A to denote the condensation of the linear order A . For instance, if $a \in A$, then $c_A(a) = \{y \in A \mid [a, y] \cap A \text{ is finite}\}$. Here are some useful properties.

LEMMA 3.12

- (1) [Rosenstein 1982, Lemma 5.14]. If \mathcal{L} is scattered and $M \subseteq L$, then $\text{FC}(\mathcal{M}) \leq \text{FC}(\mathcal{L})$.
- (2) [Rosenstein 1982, Lemma 5.13 (2)]. $\text{FC}(c^\alpha(x)) \leq \alpha$ and $c^\alpha(x)$ is a scattered interval of \mathcal{L} for every ordinal α and $x \in L$.
- (3) [Rosenstein 1982, Exercise 5.12 (1)]. If I is an interval of \mathcal{L} , then $c_I^\alpha(x) = c^\alpha(x) \cap I$ for every ordinal α and $x \in I$.
- (4) For every $x, y \in L$, if $[x, y]$ is scattered, then $c_{[x, y]}^\alpha(x) = c_{[x, y]}^\alpha(y)$ if and only if $\text{FC}([x, y]) \leq \alpha$.

PROOF. We prove the last item. Let $x, y \in L$ and α be an ordinal. Then, by definition, $\text{FC}([x, y]) \leq \alpha$ means that, (\dagger) for every $z \in [x, y]$, $c_{[x, y]}^\alpha(z) = c_{[x, y]}^{\alpha+1}(z)$, which necessarily equals $[x, y]$ since $[x, y]$ is scattered. Denote the condition $c_{[x, y]}^\alpha(x) = c_{[x, y]}^\alpha(y)$ by $(\dagger\dagger)$.

Then, (\dagger) clearly implies $(\dagger\dagger)$ by considering $z \in \{x, y\}$. For the converse, suppose $(\dagger\dagger)$. We first claim that $c_{[x, y]}^\alpha(x) = [x, y]$. Indeed, $(\dagger\dagger)$ implies that $y \in c_{[x, y]}^\alpha(x)$ since $c_{[x, y]}^\alpha$ is a condensation, which means that $[x, y]$ is a subset of the interval $c_{[x, y]}^\alpha(x)$. But also $c_{[x, y]}^\alpha(x) \subseteq [x, y]$ by item (3). Hence, $c_{[x, y]}^\alpha(x) = [x, y]$ as claimed. So if $z \in [x, y] = c_{[x, y]}^\alpha(x)$, then $c_{[x, y]}^\alpha(x) = c_{[x, y]}^\alpha(z)$ by the property of being a condensation. Hence, $z \in [x, y]$ implies that $c_{[x, y]}^\alpha(z) = [x, y]$. In particular, then also $c_{[x, y]}^{\alpha+1}(z) = [x, y]$, which implies (\dagger) as required. \square

4. RANKS OF AUTOMATIC LINEAR ORDERS

We now prove the central technical result, Theorem 4.7, via three propositions. As a matter of convenience, we introduce the following variation of VD-rank.

Definition 4.1. If \mathcal{L} is scattered, define its VD_* -rank, written $\text{VD}_*(\mathcal{L})$, as the least ordinal α such that \mathcal{L} can be written as a finite sum of orderings of VD-rank $\leq \alpha$.

For example, it is not hard to check that $\text{VD}(\omega) = \text{VD}_*(\omega) = 1$ and that $\omega 2 + 1$ has VD-rank 2 but VD_* -rank 1. We list some basic properties.

PROPERTY 4.2. Suppose \mathcal{L} is scattered.

- (1) $c^\alpha[\mathcal{L}]$ is a finite linear order if and only if $\text{VD}_*(\mathcal{L}) \leq \alpha$. So $\text{VD}_*(\mathcal{L})$ is the least ordinal such that $c^\alpha[\mathcal{L}]$ is a finite linear order.

- (2) If $M \subseteq L$, then $\text{VD}_*(\mathcal{M}) \leq \text{VD}_*(\mathcal{L})$.
- (3) $\text{VD}_*(\mathcal{L}) \leq \text{VD}(\mathcal{L}) \leq \text{VD}_*(\mathcal{L}) + 1$.
- (4) $\text{VD}_*(\mathcal{L}) = \alpha$ implies that \mathcal{L} contains an interval, say M , with $\text{VD}(\mathcal{M}) = \alpha$ and $\text{VD}_*(\mathcal{M}) = \alpha$.

PROOF. For the first item observe that for every α , $\mathcal{L} = \Sigma\{a \mid a \in c^\alpha[\mathcal{L}]\}$. Each a is an interval of the form $c^\alpha(x)$ for some $x \in L$. So, by Lemma 3.12, every a has VD-rank at most α . So, if $c^\alpha[\mathcal{L}]$ is finite, then $\text{VD}_*(\mathcal{L}) \leq \alpha$. Conversely, let $\mathcal{L} = \mathcal{L}_1 + \cdots + \mathcal{L}_k$ and $\text{VD}(\mathcal{L}_i) \leq \alpha$. Then, $L_i \subseteq c^\alpha(x)$ if $x \in L_i$. Since, for $x \in L$, the $c^\alpha(x)$ are pairwise disjoint, $c^\alpha[\mathcal{L}]$ is finite.

For the second item, suppose \mathcal{L} can be expressed as a finite sum $\mathcal{L}_1 + \cdots + \mathcal{L}_K$ where $\text{VD}(\mathcal{L}_i) \leq \alpha$. Define $M_i = M \cap L_i$. By Lemma 3.12(1), the VD-rank of M_i is at most α . But $M = M_1 + \cdots + M_k$ so $\text{VD}_*(M) \leq \alpha$.

The third follows from item (1) above and the property that $\text{VD}(\mathcal{L})$ is the least ordinal β such that $c^\beta[\mathcal{L}]$ is isomorphic to $\mathbf{1}$.

For the last item suppose $\text{VD}_*(\mathcal{L}) = \alpha$. Then \mathcal{L} can be expressed as a finite sum of orders of VD-rank (and by item (2) also VD_* -rank) at most α . There is a summand with VD_* -rank α for otherwise every summand can be written as a finite sum of linear orders of VD-rank $< \alpha$, and hence \mathcal{L} could be written as a finite sum of linear orders of VD-rank $< \alpha$, contrary to assumption. Finally by item (3) if a summand has VD_* -rank α then it has VD-rank α . \square

LEMMA 4.3. Suppose \mathcal{L} is a scattered linear order containing $\Sigma_{i \in I} A_i$ as a subordering and $\text{VD}_*(A_i) = \beta$, where I has order type ω or ω^* and each A_i is nonempty. Then $\text{VD}_*(\mathcal{L}) > \beta$.

PROOF. By Property 4.2(4) we can assume without loss of generality that $\text{VD}_*(A_i) = \text{VD}(A_i) = \beta$. This means that A_i can not be written as a finite sum of orders of VD-rank $< \beta$. Suppose that I has order type ω , the other case being similar. Let $\mathcal{A} = \Sigma_i A_i$. For every i choose some $x_i \in A_i$.

Suppose that $c_A^\beta(x_i) = c_A^\beta(x_{i+2})$ for some i . In other words, x_i is condensed to x_{i+2} in at most β steps. Then, $\beta > 0$ since $c_A^0(x) = \{x\}$ for every x . Moreover, there is some $\gamma < \beta$ so that there are finitely many elements in $c_A^\gamma[\mathcal{A}]$ between $c_A^\gamma(x_i)$ and $c_A^\gamma(x_{i+2})$. These finitely many elements are of the form $c_A^\gamma(x)$ for $x \in A$ and so have VD-rank at most γ . In particular, A_{i+1} can be written as a finite sum of orders of VD-rank at most γ , contrary to assumption. We conclude that $c_A^\beta(x_i) \neq c_A^\beta(x_{i+2})$ for every i .

Hence, $c_A^\beta[\mathcal{A}]$ is infinite and so $\text{VD}_*(\mathcal{A}) > \beta$ by Property 4.2(1). So $\text{VD}_*(\mathcal{L}) > \beta$ by Property 4.2(2). \square

PROPOSITION 4.4. Suppose \mathcal{L} is a scattered linear ordering and consider a finite partition of the domain $L = A_1 \cup A_2 \cup \cdots \cup A_k$. Then there exists some $\delta \in \{1, \dots, k\}$ with $\text{VD}_*(A_\delta) = \text{VD}_*(\mathcal{L})$.

PROOF. The proof is done by induction. So assume that $\text{VD}_*(\mathcal{L}) = \alpha$ is the ordinal to be addressed and that the proposition holds for all $\beta < \alpha$. Let L, k, A_1, \dots, A_k be as in the statement of the proposition.

If $\alpha = 0$, then $\text{VD}_*(\mathcal{L}) = \text{VD}_*(A_\epsilon) = 0$ for every nonempty subset A_ϵ of L .

Otherwise, by Property 4.2 item (4), there is some interval of \mathcal{L} , say \mathcal{M} , with $\text{VD}(\mathcal{M}) = \alpha$ and $\text{VD}_*(\mathcal{M}) = \alpha$. Then, \mathcal{M} is an \mathcal{I} -sum of linear orders $\{\mathcal{M}_i\}$ of VD -rank $< \alpha$, where \mathcal{I} is an infinite linear order of the type ω , ω^* or ζ . So suppose that \mathcal{I} is of type ω (the other two order types are similar).

Suppose α is a successor ordinal, say $\alpha = \beta + 1$. There are infinitely many i such that $\text{VD}_*(\mathcal{M}_i) = \beta$, for otherwise we could write \mathcal{M} as a finite sum of orders of VD -rank β , and conclude that $\text{VD}_*(\mathcal{M}) \leq \beta$. For each such i let $A_{\delta,i} = \mathcal{M}_i \cap A_\delta$, where $\delta \in \{1, \dots, k\}$. Applying the induction hypothesis to every \mathcal{M}_i we see that there is an $\epsilon \in \{1, \dots, k\}$ and infinitely many j such that $\text{VD}_*(\mathcal{A}_{\epsilon,j}) = \text{VD}_*(\mathcal{M}_j) = \beta$. Hence A_ϵ contains an ω -sum of linear orders of VD_* -rank β . By Lemma 4.3, $\text{VD}_*(A_\epsilon) = \alpha$.

Suppose that α is a limit ordinal. The supremum of the VD -ranks of the \mathcal{M}_i is α . Using the notation of the case above, and applying induction, we see that there is an $\epsilon \in \{1, \dots, k\}$ and infinitely many j such that $\text{VD}_*(\mathcal{A}_{\epsilon,j}) = \text{VD}_*(\mathcal{M}_j)$, and the supremum of the VD_* -ranks of these $\mathcal{A}_{\epsilon,j}$ is α . Then, $\text{VD}_*(A_\epsilon) = \alpha$ as required. \square

PROPOSITION 4.5. *Let \mathcal{L} be a scattered order with VD -rank at least α . Then, for every $\beta < \alpha$, there exists a closed interval of \mathcal{L} of VD -rank $\beta + 1$.*

PROOF. Recall that VD -ranks and FC -ranks coincide on scattered linear orders, Theorem 3.11. Fix $\beta < \alpha$. Since \mathcal{L} has FC -rank $> \beta$, by definition, there is some $x \in L$ such that $c^\beta(x) \neq c^{\beta+1}(x)$. Pick $y \in c^{\beta+1}(x) \setminus c^\beta(x)$. Then, $c^\beta(x) \neq c^\beta(y)$ and $c^{\beta+1}(x) = c^{\beta+1}(y)$. Recall that $c_{[x,y]}^\beta$ is the condensation mapping c^β within the interval $[x, y]$. Hence, by Lemma 3.12(3), $c_{[x,y]}^\beta(x) \neq c_{[x,y]}^\beta(y)$ and $c_{[x,y]}^{\beta+1}(x) = c_{[x,y]}^{\beta+1}(y)$. By Lemma 3.12(4) the first fact implies that $\text{FC}([x, y]) > \beta$ and the second fact implies that $\text{FC}([x, y]) \leq \beta + 1$. So the FC -rank of $[x, y]$ is exactly $\beta + 1$. \square

PROPOSITION 4.6. *The VD -rank of every automatic scattered linear ordering is finite.*

PROOF. Given an automatic scattered linear order \mathcal{L} over Σ^* , let $(Q_\leq, \iota_\leq, \Delta_\leq, F_\leq)$ be a deterministic 2-tape automaton recognizing the ordering of \mathcal{L} . Similarly, let $(Q_A, \iota_A, \Delta_A, F_A)$ be a deterministic 3-tape automaton recognizing the definable relation $\{(x, z, y) \mid x \leq z \leq y\}$. We assume the state sets Q_A and Q_\leq are disjoint.

For $x, y \in L$ and $v \in \Sigma^*$, define $[x, y]_v$ as the set of all $z \in L$ such that $x \leq z \leq y$ and z has prefix v . For $|v| \geq |x|, |y|$ define $I(x, v, y) \in Q_A$ and $J(x) \in Q_\leq$ as follows. $I(x, v, y)$ is the state in Q_A that results from the initial state ι_A after reading the convolution of (x, v, y) , namely $(x \diamond^n, v, y \diamond^m)$ where $n, m \geq 0$ are chosen so that the length of each component is exactly $|v|$. That is, define $I(x, v, y) := \Delta_A(\iota_A, \otimes(x, v, y))$. Similarly, define $J(v) := \Delta_\leq(\iota_\leq, \otimes(v, v))$. Write $K(x, v, y)$ for the ordered pair $(I(x, v, y), J(v))$.

Now, if $K(x, v, y) = K(x', v', y')$, then the subordering with domain $[x, y]_v$ is isomorphic to the subordering with domain $[x', y']_{v'}$ via the map $vw \mapsto v'w$ for

$w \in \Sigma^*$. Indeed the domains are isomorphic since for every $w \in \Sigma^*$,

$$vw \in [x, y]_v$$

if and only if

$$\Delta_A(\Delta_A(\iota_A, \otimes(x, v, y)), \otimes(\epsilon, w, \epsilon)) \in F_A$$

if and only if

$$\Delta_A(\Delta_A(\iota_A, \otimes(x', v', y')), \otimes(\epsilon, w, \epsilon)) \in F_A$$

if and only if

$$v'w \in [x', y']_{v'}$$

The map preserves the ordering since for $w_1, w_2 \in \Sigma^*$ such that $vw_1, vw_2 \in [x, y]_v$ and $v'w_1, v'w_2 \in [x', y']_{v'}$ we have

$$vw_1 \leq vw_2$$

if and only if

$$\Delta_{\leq}(\Delta_{\leq}(\iota_{\leq}, \otimes(v, v)), \otimes(w_1, w_2)) \in F_{\leq}$$

if and only if

$$\Delta_{\leq}(\Delta_{\leq}(\iota_{\leq}, \otimes(v', v')), \otimes(w_1, w_2)) \in F_{\leq}$$

if and only if

$$v'w_1 \leq v'w_2.$$

Hence, the number of isomorphism types of suborderings with domain $[x, y]_v$ for $|v| \geq |x|, |y|$ is bounded by the number of distinct pairs $K(x, v, y)$ which is at most $|Q_A| \times |Q_{\leq}|$, denoted by d . In particular (\dagger), there are at most d many VD*-ranks among suborderings with domain of the form $[x, y]_v$ for $|v| \geq |x|, |y|$.

Now suppose there exists a closed interval $[x, y]$ of \mathcal{L} with VD-rank at least $2(d + 2)$. Using Proposition 4.5, for every $1 \leq i \leq 2(d + 2)$, the interval $[x, y]$ contains a closed interval, say $[x_i, y_i]$, of VD-rank i . So by Property 4.2(3), at least $d + 2$ many of these intervals have different VD*-ranks; and at least $d + 1$ many of these intervals have nonzero VD*-rank. Say $[x_j, y_j]$ is one of these $d + 1$ many intervals. Set $n = \max\{|x_j|, |y_j|\}$ and partition $[x_j, y_j]$ into the set $[x_j, y_j] \cap \Sigma^{<n}$ and the finitely many sets of the form $[x_j, y_j]_v$ where $|v| = n$. Since the finite set $[x_j, y_j] \cap \Sigma^{<n}$ has VD*-rank 0, by Proposition 4.4 there is some v_j with $|v_j| = n$ so that the subordering on domain $[x_j, y_j]_{v_j}$ has the same VD*-rank as $[x_j, y_j]$. Hence there are at least $d + 1$ many intervals of the form $[x_j, y_j]_{v_j}$ all with different VD*-ranks. This contradicts (\dagger) and so we conclude that the VD-rank of every closed interval $[x, y]$ of \mathcal{L} is at most $e = 2(d + 2)$. So, for every $x, y \in L$, $c^e(x) = c^e(y)$ and so $\text{VD}(\mathcal{L}) \leq e$ as required. \square

As a corollary of the proposition just proved we derive the following result for all automatic linear orderings:

THEOREM 4.7. *The FC-rank of every automatic linear order is finite.*

PROOF. Let \mathcal{L} be a linear order and write it as $\sum\{\mathcal{L}_i \mid i \in D\}$ where D is dense and each \mathcal{L}_i is scattered. We will show that for every $i \in D$ and every $a, b \in L_i$, the VD-rank of $[a, b]$ is uniformly bounded. Let $(Q_{\leq}, \iota_{\leq}, \Delta_{\leq}, F_{\leq})$ be a deterministic 2-tape automaton recognizing the ordering of \mathcal{L} . Let $(Q_A, \iota_A, \Delta_A, F_A)$ be a deterministic 3-tape automaton recognizing the definable relation $\{(x, z, y) \mid x \leq z \leq y\}$. Now consider an interval $[a, b]$ of \mathcal{L}_i for some $i \in D$. The proof of the previous theorem ensures that the VD-rank of the scattered interval $[a, b]$ is at most e , where the constant e does not depend on $[a, b]$ or i but only on $|Q_A|$ and $|Q_{\leq}|$. Therefore, the VD-rank of interval $[a, b]$ is at most e . Hence, $\text{VD}(\mathcal{L}_i) \leq e$ for every $i \in D$ and so $\text{FC}(\mathcal{L}) \leq e$. \square

Remark 4.8. This result is a necessary though not sufficient condition for a linear order to be automatically presentable. Indeed there are linear orders of rank 2 that are not automatically presentable. For instance if $R \subset \mathbb{N}$ is a noncomputable set, then the linear order $\Sigma_{n \in R}(\zeta + \mathbf{n})$ does not have decidable first order theory and so is not automatically presentable.

COROLLARY 4.9 [DELHOMMÉ 2004]. *An ordinal α is automatically presentable if and only if $\alpha < \omega^\omega$.*

PROOF. Suppose α is an automatically presentable ordinal. Then, by Theorem 4.7, it has finite FC-rank and so, by Example 3.5, $\alpha < \omega^\omega$ as required.

Conversely, given $\alpha < \omega^\omega$, there exists $n < \omega$ such that $\alpha < \omega^n$. But ω^n is automatically presentable. Say (W, \leq) is an automatic presentation. Let $p \in W$ be the string corresponding to α . So the suborder of W on the definable domain $\{x \in W \mid x < p\}$ is isomorphic to α . Hence, α is automatically presentable. \square

PROPOSITION 4.10. *It is decidable whether or not an automatic linear order \mathcal{L} is scattered. If it is not scattered, then a regular dense subordering is effectively computable from a presentation for \mathcal{L} .*

PROOF. Let \mathcal{L} be an automatic order. The proof of Theorem 4.7 says that a bound e on the FC-rank of \mathcal{L} is computable given automata for the order and the interval relation. The condensation c_{FC} , viewed as the equivalence relation x related to y if $x \in c_{FC}(y)$, is definable in \mathcal{L} since $c_{FC}(x) = c_{FC}(y)$ if and only if $[x, y]$ is finite. Since the ordering on $c_{FC}[\mathcal{L}]$ is also definable from \mathcal{L} (see Definition 3.7) the linear orders $c_{FC}^i[\mathcal{L}]$ are definable for every $i \in \mathbb{N}$, and hence automatic. So consider $c_{FC}^e[\mathcal{L}]$. By Example 3.10, it is isomorphic to $\mathbf{1}$ if and only if \mathcal{L} is scattered. So using the decidability of the theory of $c_{FC}^e[\mathcal{L}]$, check this with the sentence $(\exists x)(\exists y)[x < y]$. In case $c_{FC}^e[\mathcal{L}]$ is not the singleton it must be an infinite dense ordering. One may view c_{FC}^e as an automatic equivalence relation on \mathcal{L} (the $c_{FC}^e(x)$ partition \mathcal{L}), and so the $<_{lex}$ -smallest representatives from every equivalence class forms a dense subordering of \mathcal{L} that is a regular subset of L . \square

5. DECIDABILITY RESULTS FOR AUTOMATIC ORDINALS

Theorem 4.7 can now be applied to prove decidability results for automatic ordinals. Contrast this with the fact that the set of computable structures that are well orderings is Π_1^1 -complete (see Rogers [1967]).

PROPOSITION 5.1. *Let $\mathcal{L} = (L, \leq)$ be an automatic structure. It is decidable whether \mathcal{L} is isomorphic to an ordinal.*

PROOF. First, check that \leq linearly orders L , by testing whether \mathcal{L} is reflexive, transitive and antisymmetric—all first-order axioms and hence computable properties. Although being a well-order is not first-order expressible (see, e.g., Theorem 13.13 in Rosenstein [1982]), the following algorithm can be used:

- (1) **Input** the presentation (L, \leq) of \mathcal{L} .
- (2) Let $D = L$.
- (3) **While** (D, \leq) is not dense and $(\forall x \in D)[\omega^* \text{ does not embed in the interval } c(x)]$
Do Replace (D, \leq) by a presentation for $c[D]$.
- (4) **End While**
- (5) If D is isomorphic to **1** then **Output** \mathcal{L} is an ordinal,
else **Output** \mathcal{L} is not an ordinal.

Every step in the algorithm is computable. Indeed, the equivalence relation on pairs (x, y) satisfying $c(x) = c(y)$ is definable as $(\neg\exists^\infty z)[x < z < y]$. So a presentation for $c[D]$ is computed by factoring D by c . The while test is expressible as

$$\neg(\forall x \neq y)(\exists z)[x < z < y]$$

and

$$(\forall x)(\neg\exists^\infty y)(c(x) = c(y) \wedge y < x).$$

The final test is expressible by $(\exists x)(\forall y)[x = y]$.

Since the FC-rank of \mathcal{L} is finite, say k , the algorithm terminates after at most $k + 1$ many while-loop tests. If \mathcal{L} is an ordinal, then $c[\mathcal{L}]$ is an ordinal and for every $x \in L$, $c(x)$ is either finite or isomorphic to ω . By induction on k , for every $0 \leq i \leq k$, $c^i[\mathcal{L}]$ passes the $(i + 1)$ 'th while-test. The resulting order $D = c^k[\mathcal{L}]$ is isomorphic to **1** as required.

If \mathcal{L} is not an ordinal, then there exists an infinite decreasing sequence of elements. Suppose there exists such a sequence $x_1 > x_2 > x_3 > \dots$ and an $n_0 \in \mathbb{N}$ such that for all $i \geq n_0$ $c(x_i) = c(x_{n_0})$. Then, the while-test fails the first time it is executed and the resulting order $D = \mathcal{L}$ is not isomorphic to the ordinal **1**. If there is no such sequence (x_i) and n_0 , then there exists a sequence, say $y_1 > y_2 > y_3 > \dots$ such that $c(y_{i+1}) \triangleleft c(y_i)$ for all $i \in \mathbb{N}$; this is an infinite decreasing sequence of elements in $c[\mathcal{L}]$. Continue inductively in this way with $c[\mathcal{L}]$ in place of \mathcal{L} . Suppose the while-test fails the m th time for some $1 \leq m \leq k$. If it fails because there is some $x \in c^{m-1}[\mathcal{L}]$ for which ω^* embeds in $c(x)$ then $D = c^{m-1}[\mathcal{L}]$ is infinite and so not isomorphic to **1**. If there is no such m , then the while-test must fail the $(k + 1)$ 'st time. In this case $D = c^k[\mathcal{L}]$ is dense but as before there is a sequence $y_1 > y_2 > y_3 > \dots$ with $c^k(y_{i+1}) \triangleleft c^k(y_i)$ for every $i \in \mathbb{N}$. In this case, D is not isomorphic to **1**. \square

We now show that the isomorphism problem for automatic ordinals is decidable. Recall that by Cantor's Normal Form Theorem if α is an ordinal then it can be uniquely decomposed as $\omega^{\alpha_1}n_1 + \omega^{\alpha_2}n_2 + \dots + \omega^{\alpha_k}n_k$, where $\alpha_1, \alpha_2, \dots, \alpha_k$ are ordinals satisfying $\alpha_1 > \alpha_2 > \dots > \alpha_k$ and k, n_1, n_2, \dots, n_k are natural numbers. The proof of deciding the isomorphism problem for automatic ordinals is

based on the fact that Cantor's normal form can be extracted from automatic presentations of ordinals.

THEOREM 5.2. *If α is an automatic ordinal, then its normal form is computable from an automatic presentation of α .*

PROOF. Let (R, \leq_{ord}) be an automatic presentation over Σ of α . Recall that the unknown ordinal is of the form $\alpha = \omega^m n_m + \omega^{m-1} n_{m-1} + \dots + \omega^2 n_2 + \omega n_1 + n_0$ where $m, n_m, n_{m-1}, \dots, n_1, n_0$ are natural numbers. Now one can compute the values m, n_0, n_1, \dots by the following algorithm:

- (1) **Input** the presentation (R, \leq_{ord}) .
- (2) Let $D = R$, $m = 0$, $n_m = 0$.
- (3) **While** $D \neq \emptyset$ **Do**
- (4) **If** D has a maximum u
Then Let $n_m = n_m + 1$, let $D = D - \{u\}$.
Else Let $L \subseteq D$ be the set of limit ordinals in D ; that is L is the set of all $x \in D$ with no immediate predecessor in D . Replace D by L , let $m = m + 1$, let $n_m = 0$.
- (5) **End While**
- (6) **Output** the formula

$$\omega^m n_m + \omega^{m-1} n_{m-1} + \dots + \omega^2 n_2 + \omega n_1 + n_0$$

using the current values of m, n_0, \dots, n_m .

Since the first-order theory of an automatic structure is decidable, each step in the algorithm is computable. Removing the maximal element from D reduces the ordinal represented by D by 1 while the corresponding n_m is increased by 1. Replacing D by the set of its limit ordinals is like dividing the ordinal represented by D by ω ; the set of limit ordinals (including 0) strictly below $\omega^m a_m + \dots + \omega^1 a_1$ has order type $\omega^{m-1} a_m + \dots + \omega^1 a_2 + a_1$. So the next coefficient can start to be computed. Based on this it is easy to verify that the algorithm computes the coefficients n_0, n_1, \dots in this order. The algorithm eventually terminates since m is bounded by the finite bound on the VD-rank of the ordinal. \square

The following is an immediate corollary.

THEOREM 5.3. *The isomorphism problem for automatic ordinals is decidable.*

Compare this with the fact that the isomorphism problem for automatic structures and even permutation structures [Blumensath 1999, compare Ishihara et al. 2002] is not decidable.

Problem 5.4. Is the isomorphism problem for automatic linear orders decidable?

6. AUTOMATIC TREE PRELIMINARIES

The remaining sections deal with trees viewed as partial orders. Theorems 7.7 and 7.10 give a necessary condition for certain trees to be automatic. The condition is similar to that for linear orders and says that the Cantor–Bendixson rank (Definition 7.1) of the tree be finite.

A tree $\mathcal{T} = (T, \preceq)$ is a partial order that has a least element r , called the root, and in which $\{y \in T \mid y \preceq x\}$ is a finite linear order for each $x \in T$. So we think of trees as growing upwards. Write $x \parallel y$ if $x \not\preceq y$ and $y \not\preceq x$. A partial order (T, \preceq) is a *forest* if there is a partition of the domain $T = \cup T_i$ such that every (T_i, \preceq) is a tree. The subtree rooted at x , written $\mathcal{T}(x)$, has domain $T(x) = \{y \in T \mid x \preceq y\}$ with order \prec restricted to this domain. The set $S(x)$ of immediate successors of x is defined as

$$S(x) = \{y \in T \mid x \prec y \wedge (\forall z)[x \preceq z \preceq y \rightarrow (z = x \vee z = y)]\}.$$

A tree \mathcal{T} is *finitely branching* if $S(x)$ is finite for each $x \in T$. A *path* of a tree (T, \preceq) is a subset $P \subseteq T$ which is linearly ordered by \preceq and maximal (under set-theoretic inclusion) with this property. Note that a path contains the root. A path with finitely many nodes is called a *finite path*; otherwise it is called an *infinite path*.

Recall that $<_{lex}$ is the length lexicographic order on Σ^* defined as $x <_{lex} y$ if either $|x| < |y|$ or $|x| = |y|$ but x lexicographic before y . For example, $\epsilon <_{lex} 0 <_{lex} 1 <_{lex} 00 <_{lex} 01 <_{lex} \dots$ in the case that $\Sigma = \{0, 1\}$. Thus, if \mathcal{T} is an automatic tree with $T \subseteq \Sigma^*$, then the length-lexicographic order on Σ^* is inherited by each set $S(x)$. This permits one to talk about the first, second, third, \dots successor of x .

7. RANKS OF AUTOMATIC TREES

Our approach to proving facts about trees is to associate a linear order with a tree, in such a way that the tree is automatic if and only if the linear order is automatic. Then, by Theorem 4.7, the linear order has finite rank which it turns out implies that the rank of the tree is finite. More precisely, in this section, it is shown that every automatic tree has finite Cantor–Bendixson Rank.

Given a tree \mathcal{T} , define a subset of T as consisting of those nodes $x \in T$ with the property that there exist at least two distinct infinite paths in the subtree of \mathcal{T} rooted at x . It follows from downward closure that this subpartial order, $d(\mathcal{T})$, is a subtree of \mathcal{T} with the same root.

For each ordinal α , define the iterated operation $d^\alpha(\mathcal{T})$ inductively as follows:

- (1) $d^0(\mathcal{T}) = \mathcal{T}$.
- (2) $d^{\alpha+1}(\mathcal{T})$ is $d(d^\alpha(\mathcal{T}))$.
- (3) If α is a limit ordinal, then $d^\alpha(\mathcal{T})$ is $\cap_{\beta < \alpha} d^\beta(\mathcal{T})$.

Definition 7.1. The *Cantor–Bendixson Rank* of a tree \mathcal{T} , written $\text{CB}(\mathcal{T})$, is the least ordinal α such that $d^\alpha(\mathcal{T}) = d^{\alpha+1}(\mathcal{T})$.

Remark 7.2. The Cantor–Bendixson Rank of an arbitrary topological space X is defined as above, using D given as $D X = \{P \in X \mid p \text{ is not isolated}\}$ instead of d . Recall that P is isolated if $\{P\}$ is an open set. So given a tree $\mathcal{T} = (T, \preceq)$, consider the following topological space. The set of elements are the infinite paths in \mathcal{T} , written $[\mathcal{T}]$. For $P \in [\mathcal{T}]$ and $x \in T$ write $x \prec P$ if $x \in P$ and say that x is on P . The basic open sets are of the form $\{P \in [\mathcal{T}] \mid x \prec P\}$ for every $x \in T$. Then, the Cantor–Bendixson Rank of this topological space, $\text{CB}[\mathcal{T}]$, is

just the least ordinal α such that $D^{\alpha+1}[\mathcal{T}] = D^\alpha[\mathcal{T}]$. Given an infinite path P of \mathcal{T} , the following statements are equivalent:

- There is a node $x \prec P$ such that P is the only infinite path of \mathcal{T} going through x ;
- $P \notin d(\mathcal{T})$;
- There is a node $x \prec P$ with $x \notin d(\mathcal{T})$.

It follows that $D[\mathcal{T}]$ consists of exactly the infinite paths of $d(\mathcal{T})$. It can be proven by transfinite induction that also

$$D^\alpha[\mathcal{T}] = [d^\alpha(\mathcal{T})].$$

Assume now that $\alpha = \text{CB}[\mathcal{T}]$. Then $d^\alpha(\mathcal{T})$ and $d^\beta(\mathcal{T})$ contain the same infinite paths for all $\beta > \alpha$, but $d^\alpha(\mathcal{T})$ might contain some nodes which are not on any infinite paths and therefore not contained in $d^{\alpha+1}(\mathcal{T})$. Thus, the two CB-ranks might differ, but they differ at most by 1:

$$\text{CB}[\mathcal{T}] \leq \text{CB}(\mathcal{T}) \leq \text{CB}[\mathcal{T}] + 1.$$

A witness \mathcal{T} with $\text{CB}[\mathcal{T}] \neq \text{CB}(\mathcal{T})$ is the tree where the domain consists of the root 0 and, for every $n > 0$, the strings $01^{a_1}01^{a_2}0\cdots 1^{a_n}0$ with $a_1 \geq a_2 \geq \cdots \geq a_n$; the ordering is the prefix-relation \preceq restricted to this domain. One has for every node $01^{a_1}01^{a_2}0\cdots 1^{a_n}0 \in \mathcal{T}$ that $01^{a_1}01^{a_2}0\cdots 1^{a_n}0 \in d^m(\mathcal{T}) \Leftrightarrow a_n \geq m$. So $d^\omega = \{0\}$. It follows that $\text{CB}[\mathcal{T}] = \omega$ by $D^\omega(\mathcal{T}) = \emptyset$ while $\text{CB}(\mathcal{T}) = \omega + 1$ by $d^{\omega+1}(\mathcal{T}) = \emptyset \neq d^\omega(\mathcal{T})$. This witness is also robust to small changes in the definition of d . If one, for example, takes $d(\mathcal{T})$ to contain exactly those nodes which are on infinitely many infinite paths of \mathcal{T} , then the resulting trees $d^\alpha(\mathcal{T})$ and derived CB-ranks are the same.

Here are some basic properties of CB-rank.

PROPERTY 7.3. *If \mathcal{T} is a countable tree with $\text{CB}(\mathcal{T}) = \alpha$ then*

- (1) *α is a countable ordinal.*
- (2) *If $d^\alpha(\mathcal{T}) \neq \emptyset$, then $d^\alpha(\mathcal{T})$ and \mathcal{T} contain uncountably many infinite paths.*
- (3) *If $d^\alpha(\mathcal{T}) = \emptyset$, then \mathcal{T} contains only countably many infinite paths. Furthermore, α is either 0 or a successor ordinal.*

PROOF. For each β , let $x_\beta \in d^\beta(\mathcal{T}) \setminus d^{\beta+1}(\mathcal{T})$. Since \mathcal{T} is countable, and $\alpha \neq \beta$ implies that $x_\alpha \neq x_\beta$, the set of ordinals β such that $d^\beta(\mathcal{T}) \setminus d^{\beta+1}(\mathcal{T}) \neq \emptyset$ is also countable. Hence, its least upper bound, a countable ordinal, say α , is $\text{CB}(\mathcal{T})$. This proves (1).

If $d^\alpha(\mathcal{T})$ is not the empty tree, then, for every $x \in d^\alpha(\mathcal{T})$ there exist $y, z \in d^\alpha(\mathcal{T})$ with $x \prec y, z$ and $y \parallel z$. In particular, the full binary tree $(\{0, 1\}^*, \preceq_p)$ embeds in $d^\alpha(\mathcal{T})$. Since $d^\alpha(\mathcal{T})$ is a subset of \mathcal{T} , the full binary tree also embeds in \mathcal{T} . This proves (2).

If $d^\alpha(\mathcal{T})$ is the empty tree, then one shows that \mathcal{T} has only countably many infinite paths as follows: For every infinite path P of \mathcal{T} , there is a minimum ordinal $\beta_P \leq \alpha$ such that $P \not\subseteq d^{\beta_P}(\mathcal{T})$. Furthermore, there is a node x_P in P such that $x_P \notin d^{\beta_P}(\mathcal{T})$. Since $x_P \in d^\gamma(\mathcal{T})$ for all $\gamma < \beta_P$, it follows that β_P

is a successor ordinal $\delta + 1$. Furthermore, P is the only infinite path of $d^\delta(\mathcal{T})$ which contains x_P . Thus, the mapping $P \rightarrow (x_P, \beta_P)$ of the infinite paths of \mathcal{T} to pairs of nodes and successor ordinals up to α is one-one. Since the range of this mapping is countable, so is its domain. Now for every $\gamma < \alpha$ the root of \mathcal{T} is in $d^\gamma(\mathcal{T})$. So if $\alpha > 0$, then it is a successor ordinal. This proves (3). \square

As a matter of convenience we introduce a variation of CB-rank.

Definition 7.4. Suppose that \mathcal{T} has countably many infinite paths. Define the CB_* -rank of \mathcal{T} , written $\text{CB}_*(\mathcal{T})$, as the least ordinal α so that $d^\alpha(\mathcal{T})$ has finitely many nodes.

This is well defined since $d^\alpha(\mathcal{T}) = \emptyset$ for some α . Note that CB_* -rank is non-increasing in the sense that, if $x \preceq y$ then $\text{CB}_*(\mathcal{T}(y)) \leq \text{CB}_*(\mathcal{T}(x))$. Also since finite trees have no infinite paths, $\text{CB}_*(\mathcal{T}) \leq \text{CB}(\mathcal{T}) \leq \text{CB}_*(\mathcal{T}) + 1$.

LEMMA 7.5. Suppose \mathcal{T} has countably many infinite paths and that \mathcal{T} is finitely branching.

- (1) $\text{CB}_*(\mathcal{T})$ is 0 or a successor ordinal.
- (2) If $\text{CB}_*(\mathcal{T}) \geq \beta + 1$, then there is some $x \in \mathcal{T}$ with $\text{CB}_*(\mathcal{T}(x)) = \beta + 1$.

PROOF. Say $\text{CB}_*(\mathcal{T}) = \alpha$ and $\alpha > 0$ is a limit ordinal. Then $d^\alpha(\mathcal{T})$ contains an infinite path as follows. The root of \mathcal{T} , call it x_0 , is in $d^\alpha(\mathcal{T})$ for otherwise $d^\gamma(\mathcal{T})$ is empty for some $\gamma < \alpha$. Since x_0 has finitely many immediate successors in \mathcal{T} , there must be one, call it x_1 , with the property that $x_1 \in d^\alpha(\mathcal{T})$ for otherwise the maximum of the CB_* -ranks of the immediate successors of x_0 is $< \alpha$ and so $\text{CB}_*(\mathcal{T}) < \alpha$. Proceed in this way to build an infinite path x_0, x_1, x_2, \dots of $d^\alpha(\mathcal{T})$. In particular then the CB_* -rank of \mathcal{T} is not α . This proves (1).

Let $\text{CB}_*(\mathcal{T}) \geq \beta + 1$. Then $d^\beta(\mathcal{T})$ is an infinite finitely branching tree and so contains some infinite path P . Moreover there must be some infinite path of $P \subseteq d^\beta(\mathcal{T})$ so that $P \not\subseteq d^{\beta+1}(\mathcal{T})$; for otherwise we could embed a copy of the infinite binary tree in $d^\beta(\mathcal{T})$ and so conclude that \mathcal{T} has uncountably many infinite paths. Hence, pick $x \in P$ with $x \notin d^{\beta+1}(\mathcal{T})$. Then $\text{CB}_*(\mathcal{T}(x)) = \beta + 1$. This proves (2). \square

For the first result, one associates the Kleene–Brouwer ordering with a tree.

Definition 7.6 (see Rogers [1967]). Let (T, \preceq) be a tree and \leq_{lex} be the length lexicographic order induced by the presentation of T as a subset of Σ^* . Let x, y be nodes on T . Define $x \leq_{kb} y$ to mean either $y \preceq x$ or there are u, v, w such that $v, w \in S(u)$, $v \preceq x$, $w \preceq y$ and $v <_{lex} w$. Write \mathcal{KB}_T for the structure (T, \leq_{kb}) .

In words, $x \leq_{kb} y$, if and only if either x is above y in the tree or x is to the left of y (with respect to $<_{lex}$ restricted to immediate successors). Note that \leq_{kb} linearly orders T and (T, \leq_{kb}) is first-order definable from (T, \preceq, \leq_{lex}) . For example, if $y_1 <_{lex} y_2 <_{lex} \dots <_{lex} y_l$ are the immediate successors of the root r of T , then $\mathcal{KB}_T = \mathcal{KB}_{T(y_1)} + \dots + \mathcal{KB}_{T(y_l)} + \mathbf{1}$. Recall that $\mathcal{T}(x)$ denotes the subtree of T with root x and that its domain is written $T(x)$.

THEOREM 7.7. *The CB-rank of an automatic finitely branching tree with countably many infinite paths is finite.*

PROOF. Suppose \mathcal{T} is finitely branching with countably many infinite paths. We now prove (\dagger) that \mathcal{KB}_T is scattered and $\text{CB}_*(\mathcal{T}) = \text{VD}_*(\mathcal{KB}_T)$. Consequently, if \mathcal{T} is automatic, then so is \mathcal{KB}_T , which by Theorem 4.7 has finite VD-rank and hence finite VD_* -rank. Then, the CB_* -rank of \mathcal{T} must be finite as required.

To prove (\dagger) proceed by induction on $\text{CB}_*(\mathcal{T})$. If \mathcal{T} has CB_* -rank 0, then \mathcal{KB}_T is finite and so has VD_* -rank 0. Before proceeding to the general case, we make some observations. Suppose \mathcal{T} contains an infinite path x_1, x_2, x_3, \dots . For a given i list $S(x_i)$ as follows: $y_1 <_{\text{llex}} \dots <_{\text{llex}} y_k <_{\text{llex}} x_{i+1} <_{\text{llex}} z_1 <_{\text{llex}} \dots <_{\text{llex}} z_l$. Define the set $L_i \subseteq \mathcal{T}$ as $\cup T(y_j)$ and define $R_i \subseteq \mathcal{T}$ as $\cup T(z_j)$. So \mathcal{L}_i and \mathcal{R}_i are forests of disjoint subtrees of \mathcal{T} . Abuse notation and define \mathcal{KB}_{L_i} as the linear order $\mathcal{KB}_{T(y_1)} + \mathcal{KB}_{T(y_2)} + \dots + \mathcal{KB}_{T(y_k)}$. Similarly define \mathcal{KB}_{R_i} as the linear order $\mathcal{KB}_{T(z_1)} + \mathcal{KB}_{T(z_2)} + \dots + \mathcal{KB}_{T(z_l)}$. Then, by definition of $<_{kb}$,

$$\mathcal{KB}_T = (\mathcal{KB}_{L_1} + \mathcal{KB}_{L_2} + \mathcal{KB}_{L_3} + \dots) + (\dots + \mathcal{KB}_{R_3} + \mathbf{1}_3 + \mathcal{KB}_{R_2} + \mathbf{1}_2 + \mathcal{KB}_{R_1} + \mathbf{1}_1), \quad (1)$$

where $\mathbf{1}_i$ has order type $\mathbf{1}$ and represents the element x_i . In particular, suppose \mathcal{T} contains exactly one infinite path. Then every \mathcal{KB}_{L_i} and \mathcal{KB}_{R_i} is a finite linear order. So depending on whether there are infinitely many i such that \mathcal{KB}_{L_i} (or \mathcal{KB}_{R_i}) is the empty linear order, \mathcal{KB}_T has one of the following scattered order types: ω^* , $\mathbf{n} + \omega^*$ for some $n \in \mathbb{N}$, or $\omega + \omega^*$. Note that these orders have VD_* -rank 1.

For the general case, suppose the CB_* -rank of \mathcal{T} is not 0. Then by Lemma 7.5(1) it is $\beta + 1$ for some ordinal β . Let $X = \{x \in \mathcal{T} \mid \text{CB}_*(\mathcal{T}(x)) = \beta + 1\}$. Then X is a downward closed subset of \mathcal{T} , and so \mathcal{X} is a tree.

The tree \mathcal{X} has infinitely many nodes. Indeed for every $x \in X$, the finitely branching tree $d^\beta(\mathcal{T}(x))$ is infinite and so contains an infinite path (w_i) . For every i the tree $\mathcal{T}(w_i)$ has CB_* -rank $\beta + 1$ and so w_i is in X . So \mathcal{X} , also being finitely branching, has at least one infinite path. Now if \mathcal{X} has infinitely many infinite paths, then, since \mathcal{X} is finitely branching, we can construct an infinite path (z_i) of \mathcal{X} such that for every i there are infinitely many infinite paths in $\mathcal{X}(z_i)$ (the subtree of \mathcal{X} with root z_i). For infinitely many i , there is a $y \in S(z_i) \setminus \{z_{i+1}\}$ with $y \in X$. So \mathcal{T} contains the infinite path (z_i) with $\text{CB}_*(\mathcal{T}(z_i)) = \beta + 1$ and for infinitely many i there is $y \in T$ that is in $S(z_i) \setminus \{z_{i+1}\}$ with $\text{CB}_*(\mathcal{T}(y)) = \beta + 1$. So $d^{\beta+1}(\mathcal{T})$ contains the infinite path (z_i) contradicting that $\text{CB}_*(\mathcal{T}) = \beta + 1$. We conclude that \mathcal{X} contains a nonzero finite number of infinite paths.

Let (x_i) be some infinite path of \mathcal{X} and define $L_i \subseteq \mathcal{T}$ and $R_i \subseteq \mathcal{T}$ as above. The forest \mathcal{L}_i (or \mathcal{R}_i) consists of finitely many disjoint subtrees of \mathcal{T} ; list these as $\mathcal{T}(w_1), \dots, \mathcal{T}(w_k)$, where $w_j \in S(x_i)$. Then $\text{CB}_*(\mathcal{T}(w_j)) \leq \beta + 1$. Moreover, since \mathcal{X} has only finitely many infinite paths, there exists $c \in \mathbb{N}$ such that for every $i \geq c$, every tree $\mathcal{T}(w_j)$ of \mathcal{L}_i and \mathcal{R}_i has CB_* -rank $\leq \beta$. By induction, $\mathcal{KB}_{T(w_j)}$ is scattered and $\text{CB}_*(\mathcal{T}(w_j)) = \text{VD}_*(\mathcal{KB}_{T(w_j)})$. So for every $i \geq c$, the linear order \mathcal{L}_i (and \mathcal{R}_i) being a finite sum of such $\mathcal{T}(w_j)$ is scattered and has VD_* -rank equal to the supremum of $\text{VD}_*(\mathcal{T}(w_1)), \dots, \text{VD}_*(\mathcal{T}(w_k))$ which is at most β . Moreover, by Lemma 7.5(2), there are infinitely many $m \geq c$ for which there exists a tree $\mathcal{T}(w_j)$ in \mathcal{L}_m (or \mathcal{R}_m) that has CB_* -rank β . We conclude that there are infinitely

many \mathcal{KB}_{L_m} (or infinitely many \mathcal{KB}_{R_m}) with VD_* -rank exactly β . Hence, using Eq. (1) and Lemma 4.3, the linear order $\mathcal{KB}_{T(x_c)}$ has VD_* -rank $\beta + 1$.

Pick $n < c$ so that \mathcal{L}_n (or \mathcal{R}_n) contains a tree $T(w_j)$ of CB_* -rank $\beta + 1$. Argue as before with $T(w_j)$ in place of T . To this end, define X' as $\{x \in T(w_j) \mid \text{CB}_*(\mathcal{T}(x)) = \beta + 1\}$. Then, as before, X' is a subtree of $T(w_j)$ with finitely many infinite paths. However, since X' does not contain the previous infinite path (x_i) , the tree X' has fewer infinite paths than X . This guarantees that after a finite number of iterations $c = 0$.

Since X has a finite (nonzero) number of infinite paths, we can write \mathcal{KB}_T as a finite (nonzero) sum of linear orders of VD_* -rank $\beta + 1$. This completes the induction. \square

THEOREM 7.8. *The CB-rank of every finitely branching automatic tree is finite.*

PROOF. Let T be a finitely branching automatic tree and \mathcal{KB}_T the Kleene–Brouwer ordering of T . Call a node $a \in T$ *scattered* if $\mathcal{T}(a)$ contains countably many infinite paths. By the proof Theorem 7.7, $\mathcal{KB}_{T(a)}$ is a scattered linear ordering and $\text{CB}_*(\mathcal{T}(a)) = \text{VD}_*(\mathcal{KB}_{T(a)})$. But applying the proof of Theorem 4.7 to the automatic linear order \mathcal{KB}_T , there exists $e \in \mathbb{N}$ such that $\text{VD}([x, y]) \leq e$ for every scattered closed interval $[x, y]$ of \mathcal{KB}_T . In particular, for every scattered $a \in T$, $\text{VD}_*(\mathcal{KB}_{T(a)}) \leq \text{VD}(\mathcal{KB}_T) \leq e$. So $\text{CB}(\mathcal{T}(a)) \leq e$ and $d^e(T)$ contains no scattered nodes.

We claim that $d^e(T) = d^{e+1}(T)$. It is always the case that $d^{e+1}(T) \subseteq d^e(T)$. If T has countably many infinite paths, then $d^e(T)$ is empty. Otherwise, suppose $x \in d^e(T)$. Then, x is not a scattered node since all the scattered nodes have been removed, and so $\mathcal{T}(x)$ contains uncountably many infinite paths. In particular, $x \in d^{e+1}(T)$. So $d^e(T) \subseteq d^{e+1}(T)$, as required. \square

Next, we remove the condition that the tree be finitely branching.

Definition 7.9. Given a tree (T, \preceq) , define a partial order $x \preceq' y$ on T by

$$x \preceq y \vee (\exists v, w \in T)[x, w \in S(v) \wedge x \leq_{lex} w \wedge w \preceq y];$$

where \leq_{lex} is the length lexicographic order and $S(v)$ the set of immediate successors of v with respect to \preceq .

Recall the set $S(x)$ is the set of \prec -immediate successors of $x \in T$. Then, since \leq_{lex} restricted to $S(x)$ has order type ω if $S(x)$ is infinite, (T, \preceq') is indeed a tree which we denote by T' . Let $s(x)$ be the length-lexicographically least element of $S(x)$ for the case $S(x) \neq \emptyset$ and let $s(x) = u$ for a default value $u \notin T$ if $S(x) = \emptyset$.

Note that \preceq' extends \preceq . For $x \in T$ let $S'(x)$ be the set of successors with respect to \preceq' . Then, $S'(x)$ contains $s(x)$ whenever $s(x) \neq u$ and the length-lexicographically next sibling y of x with respect to \preceq whenever this y exists. Recall that y is a sibling of x with respect to \preceq if there is a node z with $x, y \in S(z)$. Hence, $T' = (T, \preceq')$ is a finitely branching tree that is automatic if T is automatic.

THEOREM 7.10. *The CB-rank of an automatic tree $T = (T, \preceq)$ is finite.*

PROOF. Let U and U' be the sets of infinite paths of $\mathcal{T} = (T, \preceq)$ and $\mathcal{T}' = (T, \preceq')$, respectively. Since every infinite path of \mathcal{T} generates an infinite path of \mathcal{T}' , there is a one-one continuous mapping q from U to U' . This mapping satisfies for all $P \in U$ and all $x \in T$: $x \in P$ iff $s(x) \in q(P)$. Furthermore, U' contains besides the paths of the form $q(P)$ for some $P \in U$ also the paths generated by those sets $S(x)$ where $S(x)$ is infinite. Since there are countably many of these additional paths one has the following equivalence for all x : $\{P \in U : x \in P\}$ is uncountable iff $\{P' \in U' : s(x) \in P'\}$ is uncountable.

Now one shows by induction over n that the following implication holds for all $x \in T$ with $s(x) \neq u$ and $n \in \mathbb{N}$: $x \in d^n(\mathcal{T}) \Rightarrow s(x) \in d^n(\mathcal{T}')$. The property clearly holds for $n = 0$. Now assume the inductive hypothesis for n and consider any $x \in d^{n+1}(\mathcal{T})$. There are two distinct infinite paths $P, Q \in U$ such that $x \in P \cap Q$ and $P \cup Q \subset d^n(\mathcal{T})$. It follows that $s(x) \in q(P) \cap q(Q)$. By induction hypothesis and by q being one-one, $s(x)$ is a member of the two distinct infinite paths $q(P), q(Q)$ of $d^n(\mathcal{T}')$ and thus $s(x) \in d^{n+1}(\mathcal{T}')$. This completes the proof of this property.

By Theorem 7.8, there is a natural number n such that $d^n(\mathcal{T}')$ contains exactly those nodes of the form $s(x)$ which are in uncountably many members of U' . Then all $x \in d^n(\mathcal{T})$ satisfy that x is in uncountably many members of U . On the other hand, every x being in uncountably many members of U is in $d^n(\mathcal{T})$. So $d^n(\mathcal{T})$ contains exactly the nodes x that are in uncountably many members of U and $d^{n+1}(\mathcal{T}) = d^n(\mathcal{T})$. The CB-rank of \mathcal{T} is at most n . \square

8. AUTOMATIC VERSIONS OF KÖNIG'S LEMMA

König's Lemma says that every infinite finitely branching tree has at least one infinite path. This section consists of automatic versions of this result. If one considers Turing machines instead of finite automata there are trees that have infinite paths, but no hyperarithmetic one, and in particular no computable infinite path. Furthermore, even finitely branching trees might have infinite paths but none of them is computable. In contrast to this, the following results state that every automatic tree, not necessarily finitely branching, either has a regular infinite path or does not have an infinite path at all.

PROPOSITION 8.1. *It is decidable whether an automatic tree has an infinite path.*

PROOF. Let (T, \preceq) be an automatic tree and recall that (T, \prec_{kb}) is an automatic linear order. By Proposition 5.1, it is decidable whether this order is isomorphic to an ordinal. And this is the case if and only if (T, \preceq) has no infinite path. To prove this last statement recall that a linear order is isomorphic to an ordinal if and only if it has no infinite decreasing chain. So suppose (T, \preceq) has an infinite path $x_1 \prec x_2 \prec x_3 \dots$. Then $x_1 \succ_{kb} x_2 \succ_{kb} x_3 \dots$ is an infinite decreasing chain in (T, \leq_{kb}) , and so $(T, \leq_k b)$ is not isomorphic to an ordinal. Conversely, suppose (T, \prec_{kb}) is not isomorphic to an ordinal and let $x_1 \succ_{kb} x_2 \succ_{kb} x_3 \dots$ be an infinite decreasing chain. We define an infinite path (p_i) of (T, \prec) as follows.

- (1) Let $i = 1$ and $j = 1$.
- (2) **Repeat**

- (a) Define $p_i = x_j$.
 - (b) Replace j with the smallest $k > j$ for which there is a $u \in S(p_i)$ with $u \preceq x_l$ for every $l \geq k$.
 - (c) Replace i with $i + 1$.
- (3) **End Repeat**

If such a k exists in Step (2)(b) of every stage of the repeat loop, then the resulting sequence (p_i) is an infinite path in (T, \preceq) . So suppose that the algorithm has computed p_1, p_2, \dots, p_n with $p_1 \prec p_2 \prec \dots \prec p_n$. So $i = n$ and $j \in \mathbb{N}$. For every $m > j$ define $u(x_m)$ as the immediate successor of p_i that is $\leq x_m$. Then this sequence satisfies $u(x_m) \geq_{llex} u(x_{m+1}) \geq_{llex} u(x_{m+2}) \geq_{llex} \dots$ since $x_m >_{kb} x_{m+1} >_{kb} x_{m+2} >_{kb} \dots$. But since \leq_{llex} is isomorphic to an ordinal (of type ω) it can not have an infinite decreasing sequence. Thus, the sequence is eventually constant; that is, there is a (smallest) $k > j$ such that for every $l \geq k$ one has $u(x_k) = u(x_l) \preceq x_l$ as required. \square

8.1 Finitely Branching Automatic Trees

An infinite tree is *pruned* if every element is on some infinite path. Note that for a tree T the set of elements $E(T)$ above which there are infinitely many elements is definable as $\{x \in T \mid (\exists^\infty y)x \leq y\}$. So if T is finitely branching, then $E(T)$ consists of those nodes of T , that are on some infinite path. Indeed, if $x \in E(T)$, then by König's Lemma it is on an infinite path. Conversely, if $x \notin E(T)$ then there are only finitely many elements above it (in T) and so it is not on an infinite path. Hence, the subtree $(E(T), \preceq)$ is pruned and contains every infinite path of T . Further, if T is automatic, then so is $E(T)$.

THEOREM 8.2 (AUTOMATIC KÖNIG'S LEMMA VERSION 1). *If $T = (T, \preceq)$ is an infinite finitely branching automatic tree then it has a regular infinite path. That is, there exists a regular set $P \subseteq T$ so that P is an infinite path of T .*

PROOF. By the previous remark, replace T with the automatic pruned tree $(E(T), \preceq)$, and call the resulting tree \mathcal{T} . Recall that the length-lexicographic order $<_{llex}$ on Σ^* is automatic and therefore one can extend the presentation of \mathcal{T} to include $<_{llex}$, namely $(T, \preceq, <_{llex})$ is an automatic structure. Now define the leftmost infinite path P with respect to the length-lexicographic order of the successors of any node. P contains those nodes x for which every $y \prec x$ satisfies that $\forall z, z' \in S(y)[z \leq x \Rightarrow z <_{llex} z']$, and so by Proposition 2.3 P is regular. This means, that the unique node $z \in S(y)$, which is below x , is just the length-lexicographically least element of $S(y)$. Since the length-lexicographic ordering of Σ^* is a well-ordering (of type ω), this minimum always exists.

We briefly check that P is an infinite path. First P is closed downward. Indeed, given $x \in P$, let $a \preceq x$. Then for every $y \prec a$, if $z, z' \in S(y)$ and $z \leq y \preceq x$ so by hypothesis then $z \leq_{llex} z'$, as required. Second P is linearly ordered. For otherwise, if $x, a \in P$ with $x \parallel a$, then let z be their \prec -maximal common ancestor. Consider two successors of z say v and w with $v \prec x$ and $w \prec a$. Without loss of generality, suppose that $v <_{llex} w$. Then, z, v and w form a counterexample to a 's membership in P . Finally, P is infinite (and hence maximal with these properties). Indeed, if $x \in P$, then the $<_{llex}$ -smallest element in $S(x)$ is also in P . Hence, P is an infinite regular path in \mathcal{T} , as required. \square

If in the hypothesis above \mathcal{T} contains finitely many infinite paths, then every infinite path is regular since after defining P , one considers the tree on domain $\mathcal{T} \setminus P$ to find the next infinite path. The next theorem generalizes this to the case when \mathcal{T} contains countably many infinite paths.

THEOREM 8.3 (AUTOMATIC KÖNIG'S LEMMA VERSION 2). *If \mathcal{T} is an automatic tree that is finitely branching and has countably many infinite paths, then every infinite path in it is regular.*

PROOF. As before replace \mathcal{T} with the automatic pruned tree $(E(\mathcal{T}), \leq)$. Then, the derivate $d(\mathcal{T})$ is definable and so the elements of the tree $\mathcal{T} \setminus d(\mathcal{T})$ form a regular subset of \mathcal{T} , call it R . Then, R consists of countably many disjoint infinite paths, each definable as follows. For every \prec -minimal $a \in R$, define the infinite path P_a as $\{x \in T \mid x \leq a \vee (a \prec x \wedge x \in R)\}$.

Now replace \mathcal{T} by $d(\mathcal{T})$ and repeat the steps in the previous paragraph. Since $\text{CB}(\mathcal{T})$ is finite, these steps can be iterated at most $\text{CB}(\mathcal{T})$ times; after which time the resulting tree will be empty and every infinite path in the original \mathcal{T} will have been generated at some stage. \square

Remark 8.4. The assumption that \mathcal{T} have countably many infinite paths can not be dropped, since otherwise \mathcal{T} necessarily has non-regular (indeed, uncountably many noncomputable) infinite paths.

8.2 The General Case

It turns out that automaticity allows one to remove the condition that \mathcal{T} be finitely branching, under the assumption of course that \mathcal{T} has at least one infinite path. This can be done if given an automatic tree \mathcal{T} , one can effectively construct an automatic copy of the pruned tree $E(\mathcal{T})$, the set of elements of \mathcal{T} that are on an infinite path in \mathcal{T} . Then, as in the finitely branching case, Theorem 8.2, the \prec_{lex} -least path is definable and hence regular.

THEOREM 8.5 (AUTOMATIC KÖNIG'S LEMMA VERSION 3). *If \mathcal{T} is an automatic tree with an infinite path, then it has a regular infinite path.*

This follows immediately from the following construction.

LEMMA 8.6. *If \mathcal{T} is an automatic tree, then $E(\mathcal{T}) \subseteq T$ is a regular language.*

PROOF. Let $\mathcal{T} = (T, \leq)$ be an automatic tree. Writing T' for $E(\mathcal{T})$, it is required that the set $T' \subseteq \Sigma^*$ of all nodes in T that are on an infinite path is a regular language.

The idea is to construct a Büchi recognizable language \mathcal{B} over the alphabet $\Delta = \Sigma_\diamond \times \Sigma$ so that its projection (on the first co-ordinate) is of the form $T' \cdot \{\diamond\} \cdot W^\omega$ for some regular $W \subseteq \Sigma_\diamond^*$. Then, T' is regular since Büchi automata are closed under projection and an automaton for T' can be extracted from one for \mathcal{B} .

Say that a word x is on $c_0c_1\dots$, where each c_i is $(a_i, b_i) \in \Sigma_\diamond \times \Sigma$, iff there exist $m, n \in \mathbb{N}$ such that

- either $m = 0$, $x = a_0a_1\dots a_n$ and $a_{n+1} = \diamond$
- or $n \geq m > 0$, $x = b_0b_1\dots b_{m-1}a_ma_{m+1}\dots a_n$, $a_{m-1} = \diamond$ and $a_{n+1} = \diamond$.

In the first case, we say that x is the *first word* on $c_0c_1\dots$. Consider the set of all sequences $(a_0, b_0)(a_1, b_1)\dots \in \Delta$ such that there are infinitely many words on the sequence and the words on the sequence generate an infinite path of T . More formally,

- $\exists^\infty n(a_n = \diamond)$;
- if y, z are on $(a_0, b_0)(a_1, b_1)\dots$ and $|y| \leq |z|$ then $y \preceq z$ and $y, z \in T$.

There is a Büchi automaton \mathcal{B} accepting such sequences because the orderings \preceq and length-comparison are automatic and T is regular. Further, using that T is transitive, one need only check that adjacent words y, z on the sequence satisfy $y \preceq z$.

To complete the proof, we prove that $x \in T'$ if and only if x is the first word on some sequence $c_0c_1\dots$ satisfying the two conditions. The reverse implication is clear. For the forward implication, let $x \in T$ be given and P be an infinite path witnessing that $x \in T'$. Define the sequences $a_0a_1\dots$ and $b_0b_1\dots$ described below.

- (1) **Choose** n, a_0, a_1, \dots, a_n such that $x = a_0a_1\dots a_n$. **Let** $a_{n+1} = \diamond$.
- (2) **Let** $m = 0$. **Let** $y = x$.
- (3) **Find** $b_m b_{m+1} \dots b_{n+1}$ such that infinitely many nodes in P extend $b_0 b_1 \dots b_{m-1}$ as strings.
- (4) **Update** $m = n + 2$.
- (5) **Find** a new value for n and $a_m a_{m+1} \dots a_n$ such that $n \geq m$, the path P contains the node $z = b_0 b_1 \dots b_{m-1} a_m a_{m+1} \dots a_n$ and $y \preceq z$. **Let** $a_{n+1} = \diamond$.
- (6) **Let** $y = z$. **Go to** 3.

Note that it is an invariant of the construction that whenever the algorithm comes to Step (3), either $m = 0$ or infinitely many nodes in P extend the string $b_0 b_1 \dots b_{m-1}$. As there are only finitely many choices for the new part $b_m b_{m+1} \dots b_{n+1}$, one can choose this part such that still infinitely many nodes in P extend $b_0 b_1 \dots b_{n+1}$ as a string. In Step(4), m is chosen such that the precondition of Step (3) holds again and b_m is the first of the b -symbols not yet defined. For every $y \in P$, it holds that all but finitely many nodes z in P satisfy $y \preceq z$. Furthermore, for every finite length l , almost all nodes in P are represented by strings longer than l . Thus, one can find a node z as specified in Step (5) and the algorithm runs forever defining the infinite sequence $(a_0, b_0)(a_1, b_1)\dots$ in the limit. In particular, such a sequence exists. It is not required that the sequence can be constructed effectively since the path P might not even be computable. \square

From Theorem 8.5, we see that if an automatic tree has *finitely* many infinite paths, then each is regular. The next theorem generalizes this to trees with *countably* many infinite paths.

THEOREM 8.7 (AUTOMATIC KÖNIG'S LEMMA VERSION 4). *Every infinite path in an automatic tree with countably many infinite paths is regular.*

PROOF. Let $\mathcal{T} = (T, \preceq)$ be an automatic tree with countably many infinite paths. Then, the extendible part of \mathcal{T} , $E(\mathcal{T}) \subseteq T$, is regular by Lemma 8.6. So

the derivative $d(\mathcal{T})$ is automatic. Write $E^i(\mathcal{T}) \subseteq T$ for the extendible part of the domain of $d^i(\mathcal{T})$. Then, since \mathcal{T} is automatic $\text{CB}(\mathcal{T})$ is finite, say n . And since \mathcal{T} has countably many infinite paths, $d^n(\mathcal{T})$ is the empty tree. So the structure $(T, E^0(\mathcal{T}), E^1(\mathcal{T}), \dots, E^n(\mathcal{T}), \preceq)$ is automatic.

Now, for every $x \in \mathcal{T}$, there exists an $m < n$ such that x is in the domain the tree $d^m(\mathcal{T})$ and not in the domain of the tree $d^{m+1}(\mathcal{T})$. In particular, if P is an infinite path of \mathcal{T} , then there is a largest $m < n$ such that $P \subseteq E^m(\mathcal{T})$. The path P is isolated on $(E^m(\mathcal{T}), \preceq)$ since otherwise P would also be an infinite path of $d^{m+1}(\mathcal{T})$ and a subset of $E^{m+1}(\mathcal{T})$. Define $x_P \in P$ to be the least, with respect to \preceq , element of P , which is not in $E^{m+1}(\mathcal{T})$. Then P is the only infinite path of $E^m(\mathcal{T})$ containing x_P . So P is the set of all $y \in E^m(\mathcal{T})$, which are comparable to x_P with respect to \preceq . Hence, P is regular. \square

Remark 8.8. If $\mathcal{T} = (T, \preceq)$ is an automatic tree with countably many infinite paths, then there is a formula specifying all these paths that is built from the parameters $n, E^0(\mathcal{T}), E^1(\mathcal{T}), \dots, E^n(\mathcal{T})$ defined in the previous proof. The formula is the following one:

$$\begin{aligned}\Phi(a, b) = \bigvee_{i=0}^{n-1} [a \in E^i(\mathcal{T}) \wedge a \notin E^{i+1}(\mathcal{T}) \wedge b \in E^i(\mathcal{T})] \\ \wedge [(b \preceq a) \vee (a \prec b \wedge (\forall c, d, e \in E^i(\mathcal{T})) \\ \times [a \preceq c \wedge d, e \in S(c) \wedge d \preceq b \Rightarrow d \leq_{\text{llex}} e])].\end{aligned}$$

The formula Φ and each set P_a defined as $\{b \in T \mid \Phi(a, b)\}$ satisfy the following conditions:

- If $a \in E^0$, then P_a is an infinite path of \mathcal{T} ;
- If $a \notin E^0$, then P_a is empty;
- For every infinite path P of \mathcal{T} , there is an a with $P = P_a$.

ACKNOWLEDGMENTS

The authors thank Wolfgang Merkle for useful discussions and the referee for valuable comments and suggestions.

REFERENCES

- BLUMENSATH, A. 1999. Automatic structures. Diploma Thesis, RWTH Aachen.
- BLUMENSATH, A. AND GRÄDEL, E. 2000. Automatic structures. In *Proceedings of the 15th Symposium on Logic in Computer Science (LICS2000)*. IEEE Computer Society Press, Los Alamitos, CA, 51–62.
- DELHOMMÉ, C. 2004. Automaticité des ordinaux et des graphes homogènes. *Comput. Rend. Math.* 339, 1, 5–10.
- DELHOMMÉ, C. 2004. Automaticité des ordinaux et des graphes homogènes. *C.R. Acad. Sci. Paris Ser. I* 339, 5–10.
- EILENBERG, S., ELGOT, C., AND SHEPHERDSON, J. 1969. Sets recognized by n -tape automata. *J. Alg.* 13, 4, 447–464.
- EPSTEIN, D. B. H., CANNON, J. W., HOLT, D. F., LEVY, S. V. F., PATERSON, M. S., AND THURSTON, W. P. 1992. *Word processing in groups*. Jones and Bartlett.
- HAUSDORFF, F. 1908. Grundzüge einer theorie der geordnete mengen. *Math. Ann.* 65, 435–505.

- ISHIHARA, H., KHOUSSAINOV, B., AND RUBIN, S. 2002. Some results on automatic structures. In *Proceedings of the 17th Symposium on Logic in Computer Science (LICS2002)*. IEEE Computer Society Press, Los Alamitos, CA, 235–242.
- KECHRIS, A. 1995. *Classical Descriptive Set Theory*. Springer-Verlag, New York.
- KHOUSSAINOV, B. AND NERODE, A. 1994. Automatic presentations of structures. In *Logic and Computational Complexity*, International Workshop LCC 1994 (Indianapolis, IN), D. Leivant, Ed. Lecture Notes in Computer Science, vol. 960. Springer-Verlag, New York, 367–392.
- KHOUSSAINOV, B. AND NERODE, A. 2001. *Automata theory and its applications*. Progress in Computer Science and Applied Logic, vol. 21. Birkhäuser Boston Inc., Boston, MA.
- KHOUSSAINOV, B., NIES, A., RUBIN, S., AND STEPHAN, F. 2004a. Automatic structures: Richness and limitations. In *Proceedings of the 19th Annual IEEE Symposium on Logic in Computer Science (LICS)*. IEEE Computer Society Press, Los Alamitos, CA, 44–53.
- KHOUSSAINOV, B. AND RUBIN, S. 2001. Graphs with automatic presentations over a unary alphabet. *J. Automata, Lang. Combinat.* 6, 4, 467–480.
- KHOUSSAINOV, B. AND RUBIN, S. 2003. Automatic structures—Overview and future directions. *J. Automata, Lang. Combinat.* 8, 2, 287–301.
- KHOUSSAINOV, B., RUBIN, S., AND STEPHAN, F. 2004b. Definability and regularity in automatic structures. In *Proceedings of the 21st Symposium on Theoretical Aspects of Computer Science (STACS 2004)*. 440–451.
- KUSKE, D. 2003. Is Cantor’s theorem automatic ? In *Proceedings of the 10th International Conference on Logic for Programming Artificial Intelligence and Reasoning (LPAR 2003)*. 330–343.
- LOHREY, M. 2003. Automatic structures of bounded degree. In *Proceedings of the 10th International Conference on Logic for Programming Artificial Intelligence and Reasoning (LPAR 2003)*. 344–358.
- ROGERS, H. 1967. *Theory of Recursive Functions and Effective Computability*. McGraw-Hill, New York.
- ROSENSTEIN, J. 1982. *Linear Orderings*. Academic Press, Orlando, FL.

Received November 2003; revised June 2004 and August 2004; accepted August 2004

AUTOMATIC STRUCTURES: RICHNESS AND LIMITATIONS

BAKHADYR KHOUSSAINOV^a, ANDRÉ NIES^b, SASHA RUBIN^c, AND FRANK STEPHAN^d

^{a,b,c} Department of Computer Science, University of Auckland, New Zealand
e-mail address: {bmk, andre, rubin}@cs.auckland.ac.nz

^d F. Stephan, School of Computing and Department of Mathematics, National University of Singapore, Republic of Singapore
e-mail address: fstephan@comp.nus.edu.sg

ABSTRACT. We study the existence of automatic presentations for various algebraic structures. An automatic presentation of a structure is a description of the universe of the structure by a regular set of words, and the interpretation of the relations by synchronised automata. Our first topic concerns characterising classes of automatic structures. We supply a characterisation of the automatic Boolean algebras, and it is proven that the free Abelian group of infinite rank, as well as certain Fraïssé limits, do not have automatic presentations. In particular, the countably infinite random graph and the random partial order do not have automatic presentations. Furthermore, no infinite integral domain is automatic. Our second topic is the isomorphism problem. We prove that the complexity of the isomorphism problem for the class of all automatic structures is Σ_1^1 -complete.

1. INTRODUCTION

Classes of infinite structures with nice algorithmic properties (such as decidable model checking) are of increasing interest in a variety of fields of computer science. For instance the theory of infinite state transition systems concerns questions of symbolic representations, model checking, specification and verification. Also, string query languages in databases may be captured by (decidable) infinite string structures. In these and other areas there has been an effort to extend the framework of finite model theory to infinite models that have finite presentations.

Automatic structures are (usually) infinite relational structures whose domain and atomic relations can be recognised by finite automata operating synchronously on their

2000 ACM Subject Classification: F.1.1, F.4.3,

Key words and phrases: Automatic structures, automatic presentation, analytical hierarchy, isomorphism problem, complexity, finite automaton, formal languages.

^{a,b} The first and the second authors' research was partially supported by the Marsden Fund of New Zealand.

^c S. Rubin is supported by a New Zealand Science and Technology Post-Doctoral Fellowship.

^d F. Stephan previously worked at National ICT Australia which is funded by the Australian Government's Department of Communications, Information Technology and the Arts and the Australian Research Council through Backing Australia's Ability and the ICT Centre of Excellence Program. Currently, F. Stephan is supported in part by NUS grant number R252-000-212-112.

input. Consequently, automatic structures have finite presentations and are closed under first order interpretability (as well as some of extensions of first order logic). Moreover, the model checking problem for automatic structures is decidable. Hence automatic structures and tools developed for their study are well suited to these fields of computer science, see for instance [2].

From a computability and logical point of view, automatic structures are used to provide generic examples of structures with decidable theories, to investigate the relationship between automata and definability, and to refine the ideas and approaches in the theory of computable structures. This paper investigates the problem of characterising automatic structures in algebraic, model theoretic or logical terms.

Specifically this paper addresses two foundational problems in the theory of automatic structures. The first is that of providing *structure theorems* for classes of automatic structures. Fix a class \mathcal{C} of structures (over a given signature), closed under isomorphism. For instance, \mathcal{C} may be the class of groups (G, \cdot) or the class of linear orders (L, \leq) . A structure theorem should be able to distinguish whether a given member of \mathcal{C} has an automatic presentation or not (a special case of this is telling whether a given structure is automatically presentable or not). This usually concerns the interactions between the combinatorics of the finite automata presenting the structures and properties of the structures themselves. The second problem, which is related to the first, is the complexity of the *isomorphism problem* for classes of automatic structures. Namely, fix a class of automatic structures \mathcal{C} . The isomorphism problem asks, given automatic presentations of two structures from \mathcal{C} , are the structures isomorphic?

With regard to the first problem, we provide new techniques for proving that some foundational structures in computer science and mathematics do not have automatic presentations. For example, we show that the Fraïssé limits of many classes of finite structures, such as finite partial orders or finite graphs, do not have automatic presentations. This shows that the infinite random graph and random partial order do not have automatic presentations. The idea is that the finite amount of memory intrinsic to the finite automata presenting the structure can be used to extract algebraic and model theoretic properties (invariants) of the structure, and so used to classify such structures up to isomorphism. This line of research has indeed been successful in investigating automatic ordinals, linear orders, trees and Boolean algebras. For example there is a full structure theorem for the automatically presentable ordinals; namely, they are those strictly less than ω^ω [7]. There are also partial structure theorems saying that automatic linear orders and automatic trees have finite Cantor-Bendixson rank [13]. In this paper we provide a structure theorem for the (infinite) automatic Boolean algebras; namely, they are those isomorphic to finite products of the Boolean algebra of finite and co-finite subsets of \mathbb{N} .

With regard to the second problem, it is not surprising that the isomorphism problem for the class of all automatic structures is undecidable [5]. The reason for the undecidability is that the configuration space of a Turing machine, considered as a graph, is an automatic structure. Moreover, the reachability problem in the configuration space is undecidable. Thus with some extra work, as in [4] or [11], one can reduce the reachability problem to the isomorphism problem for automatic structures. In addition, the isomorphism problem for automatic ordinals [13] and Boolean algebras (Corollary 3.5) is decidable. For equivalence structures it is in Π_1^0 (though not more is known), and for configuration spaces of Turing machines is Π_3^0 -complete [16].

Hence it is somewhat unexpected that the complexity of the isomorphism problem for the class of automatic structures is Σ_1^1 -complete. The Σ_1^1 -completeness is proved by reducing the isomorphism problem for computable trees, known to be Σ_1^1 -complete [9], to the isomorphism problem for automatic structures.

The two problems are related in the following way. If one has a ‘nice’ structure theorem for a class \mathcal{C} of automatic structures, then one expects that the isomorphism problem for \mathcal{C} be computationally ‘reasonable’. For instance, as corollaries of the structure theorems for automatic ordinals and for automatic Boolean algebras, one obtains that their corresponding isomorphism problems are decidable. In contrast, the Σ_1^1 -completeness of the isomorphism problem of the class of all automatic structures tells us that the language of first order arithmetic is not powerful enough to give a structure theorem for the class of all automatic structures. In other words we should not expect a ‘nice’ structure theorem for the class of all automatic structures.

Here is an outline of the rest of the paper. The next section is a brief introduction to the basic definitions. Section 3 provides some counting techniques sufficient to prove non-automaticity of many classical structures such as fields, integral domains and Boolean algebras. Section 4 provides a technique that is used to show non-automaticity of several structures, such as the infinite random graph and the random partial order. The last section is devoted to proving that the isomorphism problem for automatic structures is Σ_1^1 -complete.

Finally, we note that automatic structures can be generalised in several directions: for instance, by using finite automata on infinite strings [4, 5], finite ranked trees [4, 1], or finite unranked trees [14]. Although this paper only deals with the finite word case, some of the ideas presented here should give rise to a better understanding of the generalisations. Indeed, Delhommé, who independently proved that the random graph has no automatic presentation (using a similar technique as the one presented here) has extended the technique to tree-automatic structures [7]. The authors would like to thank referees for comments on improvement of this paper.

2. PRELIMINARIES

A thorough introduction to automatic structures can be found in [4] and [12]. We assume familiarity with the basics of finite automata theory though to fix notation the necessary definitions are included. A *finite automaton* \mathcal{A} over an alphabet Σ is a tuple (S, ι, Δ, F) , where S is a finite set of *states*, $\iota \in S$ is the *initial state*, $\Delta \subset S \times \Sigma \times S$ is the *transition table* and $F \subset S$ is the set of *final states*. A *computation* of \mathcal{A} on a word $\sigma_1\sigma_2\dots\sigma_n$ ($\sigma_i \in \Sigma$) is a sequence of states, say q_0, q_1, \dots, q_n , such that $q_0 = \iota$ and $(q_i, \sigma_{i+1}, q_{i+1}) \in \Delta$ for all $i \in \{0, 1, \dots, n-1\}$. If $q_n \in F$, then the computation is *successful* and we say that automaton \mathcal{A} *accepts* the word. The *language* accepted by the automaton \mathcal{A} is the set of all words accepted by \mathcal{A} . In general, $D \subset \Sigma^*$ is *finite automaton recognisable*, or *regular*, if D is the language accepted by a finite automaton \mathcal{A} .

The following definitions extends recognisability to relations of arity n , called *synchronous n -tape automata*. A synchronous n -tape automaton can be thought of as a one-way Turing machine with n input tapes [8]. Each tape is regarded as semi-infinite having written on it a word in the alphabet Σ followed by an infinite succession of blanks, \diamond symbols. The automaton starts in the initial state, reads simultaneously the first symbol of each tape, changes state, reads simultaneously the second symbol of each tape, changes state, etc., until it reads a blank on each tape. The automaton then stops and accepts the n -tuple

of words if it is in a final state. The set of all n -tuples accepted by the automaton is the relation recognised by the automaton. Here is a definition.

Definition 2.1. Write Σ_\diamond for $\Sigma \cup \{\diamond\}$ where \diamond is a symbol not in Σ . The *convolution of a tuple* $(w_1, \dots, w_n) \in \Sigma^{*n}$ is the string $\otimes(w_1, \dots, w_n)$ of length $\max_i |w_i|$ over alphabet $(\Sigma_\diamond)^n$ defined as follows. Its k 'th symbol is $(\sigma_1, \dots, \sigma_n)$ where σ_i is the k 'th symbol of w_i if $k \leq |w_i|$ and \diamond otherwise.

The *convolution of a relation* $R \subset \Sigma^{*n}$ is the relation $\otimes R \subset (\Sigma_\diamond)^{n*}$ formed as the set of convolutions of all the tuples in R . That is $\otimes R = \{\otimes w \mid w \in R\}$.

Definition 2.2. An *n -tape automaton* on Σ is a finite automaton over the alphabet $(\Sigma_\diamond)^n$. An n -ary relation $R \subset \Sigma^{*n}$ is *finite automaton recognisable* (in short FA recognisable) or *regular* if its convolution $\otimes R$ is recognisable by an n -tape automaton.

We now relate n -tape automata to structures. A *structure* \mathcal{A} consists of a countable set A called the *domain* and some relations and operations on A . We may assume that \mathcal{A} only contains relational predicates as the operations can be replaced with their graphs. We write $\mathcal{A} = (A, R_1^A, \dots, R_k^A, \dots)$ where R_i^A is an n_i -ary relation on A . The relation R_i are sometimes called basic or atomic relations. We assume that the function $i \rightarrow n_i$ is always a computable one.

Definition 2.3. A structure \mathcal{A} is *automatic* over Σ if its domain $A \subset \Sigma^*$ is finite automata recognisable, and there is an algorithm that for each i produces a finite automaton recognising the relation $R_i^A \subset (\Sigma^*)^{n_i}$. A structure is called *automatic* if it is automatic over some alphabet. If \mathcal{B} is isomorphic to an automatic structure \mathcal{A} , then call \mathcal{A} an *automatic presentation* of \mathcal{B} and say that \mathcal{B} is called *automatically presentable* (over Σ).

An example of an automatic structure is the word structure $(\{0, 1\}^*, L, R, E, \preceq)$, where for all $x, y \in \{0, 1\}^*$, $L(x) = x0$, $R(x) = x1$, $E(x, y)$ iff $|x| = |y|$, and \preceq is the lexicographical order. The configuration graph of any Turing machine is another example of an automatic structure. Examples of automatically presentable structures are $(\mathbb{N}, +)$, (\mathbb{N}, \leq) , (\mathbb{N}, S) , the group $(\mathbb{Z}, +)$, the order on the rationals (Q, \leq) , and the Boolean algebra of finite or co-finite subsets of \mathbb{N} . Note that every finite structure is automatically presentable. We use the following important theorem without reference.

Theorem 2.1. [12] Let \mathcal{A} be an automatic structure. There exists an algorithm that from a first order definition (with possible use of the additional quantifier ‘there exists infinitely many’) in \mathcal{A} of a relation R produces an automaton recognising R . \square

3. PROVING NON-AUTOMATICITY VIA COUNTING

The first technique for proving non-automaticity was presented in [12] and later generalised in [4]. The technique is based on a pumping argument and exhibits the interplay between finitely generated (sub) algebras and finite automata. We briefly recall the technique for completeness.

A relation $R \subset (\Sigma^*)^n$ is called *locally finite* if there exists k, l with $k + l = n$ so that for every \bar{a} (of size k) there are at most a finite number of \bar{b} (of size l) such that $(\bar{a}, \bar{b}) \in R$. We write $R \subset (\Sigma^*)^{k+l}$. For $\bar{b} = (b_1, \dots, b_m)$, write $b \in \bar{b}$ to mean $b = b_i$ for some i .

We start with the following elementary but important proposition.

Proposition 3.1. *Let $R \subset (\Sigma^*)^{k+l}$ be locally finite, with k and l as in the definition above. Suppose further that R is a regular relation, and that the automaton for $\otimes(R)$ has p states. Then*

$$\max\{|y| \mid y \in \bar{y}\} - \max\{|x| \mid x \in \bar{x}\} \leq p$$

for every $(\bar{x}, \bar{y}) \in R$ where \bar{x} has k elements and \bar{y} has l elements.

Proof. Fix $(\bar{x}, \bar{y}) \in R$ and say $x' \in \bar{x}$ has length $\max\{|x| \mid x \in \bar{x}\}$ and say $y' \in \bar{y}$ has length $\max\{|y| \mid y \in \bar{y}\}$. So $|y'| - |x'| > p$ implies that we can pump the string $\otimes(\bar{x}, \bar{y})$ between positions $|x'|$ and $|y'|$. Then either the automaton for $\otimes(R)$ accepts a string that is not in $\otimes((\Sigma^*)^n)$ (because one of the components contains a subword of the form $\diamond\Sigma$) or otherwise it accepts strings of the form $\otimes(\bar{x}, \bar{z})$ for infinitely many \bar{z} , contradicting that R is locally finite. \square

The typical application of this proposition is to prove that certain structures do not have automatic presentations. Assume \mathcal{A} is an automatic structure in which each atomic relation R_i is a graph of a function f_i , $i = 1, \dots, m$. Let a_1, a_2, \dots be a sequence of some elements of A such that the relation $\{(a_i, a_j) \mid j = i + 1\}$ is regular. Consider the sequence $G_1 = \{a_1\}$, $G_{n+1} = G_n \cup \{a_{n+1}\} \cup \{f_i(\bar{a}) \mid \bar{a} \in G_n, i = 1, \dots, m\}$. By the proposition there is a constant p such that the length of all elements in G_n is bounded by $p \cdot n$. Therefore the number of elements in G_n is bounded by $2^{O(n)}$. Some combinatorial reasoning combined with this observation can now be applied to provide examples of structures with no automatic presentations, see [4] and [12]. For example, the following structures have no automatic presentation:

- (1) The free group on $k > 1$ generators;
- (2) The structure $(\mathbb{N}, |)$;
- (3) The structure (\mathbb{N}, p) , where $p : \mathbb{N}^2 \rightarrow \mathbb{N}$ is a bijection;
- (4) The term algebra generated by finitely many constants with at least one non-unary atomic operation¹.

Note that each of these structures has a decidable first order theory.

In the next sections we provide other more intricate techniques for showing that particular structures do not have automatic presentations. We then apply those techniques to give a characterisation of Boolean algebras that have automatic presentations. We also prove that (\mathbb{Q}^+, \times) has no automatic presentation and show that no infinite integral domain (in particular no infinite field) has an automatic presentation. We also study automaticity of some Fraïssé limits.

First we introduce a very useful property true of every automatic monoid (M, \times) .

Lemma 3.2. *For each $s_1, \dots, s_m \in M$, $|\prod_i s_i| \leq \max_i |s_i| + k \lceil \log m \rceil$, where k is the number of states in the automaton recognising the graph of \times .*

Proof. Here logarithm is to base 2 and $\lceil \log n \rceil$ is the least i such that $2^i \geq n$. We use induction on m . For $m = 1$, the inequality becomes $|s_1| \leq |s_1|$. If $m > 1$ let $m = u + v$ where $u = \lfloor m/2 \rfloor$. Apply Proposition 3.1 to the graph of the monoid operation \times and elements $x = \prod_{i=1}^u s_i$ and $y = \prod_{i=u+1}^m s_i$. Then, by induction, $|\prod_i s_i| \leq k + \max(|x|, |y|)$ which is equal to

$$k + \max\left(\max_{1 \leq i \leq u} |s_i| + k \lceil \log u \rceil, \max_{u+1 \leq i \leq m} |s_i| + k \lceil \log v \rceil\right),$$

¹Thus, elements of the term algebra are all the ground terms, and the operations are defined in a natural way: the value of a function f of arity n from the language on ground terms g_1, \dots, g_n is $f(g_1, \dots, g_n)$.

which is at most $\max_i |s_i| + k \lceil \log m \rceil$, since $1 + \max(\lceil \log u \rceil, \lceil \log v \rceil) \leq \lceil \log m \rceil$. \square

3.1. Automatic Boolean algebras. All finite Boolean algebras are automatic. Thus, in this section we deal with infinite countable Boolean algebras only. Our goal in this section is to give a full characterisation of infinite automatic Boolean algebras. Our characterisation can then be applied to show that the isomorphism problem for automatic Boolean algebras is decidable. Compare this with the result from computable algebra that the isomorphism problem for computable Boolean algebras is Σ_1^1 -complete [9].

Recall that a Boolean algebra $\mathcal{B} = (B, \cup, \cap, \setminus, \mathbf{0}, \mathbf{1})$ is a structure, where \cap and \cup and \setminus operations satisfy all the basic properties of the set-theoretic intersection, union, and complementation operations; In \mathcal{B} the relation $a \subseteq b \iff a \cap b = a$ is a partial order in which $\mathbf{0}$ is the smallest element, and $\mathbf{1}$ is the greatest element. The complement of an element $b \in B$ is $\mathbf{1} \setminus b$ and is denoted by \bar{b} .

A linearly ordered set determines a Boolean algebra in a natural way described as follows. Let $\mathcal{L} = (L, \leq)$ be a linearly ordered set with a least element. An interval is a subset of L of the form $[a, b] = \{x \mid a \leq x < b\}$, where $a, b \in L \cup \{\infty\}$. The **interval algebra** denoted by $\mathcal{B}_{\mathcal{L}}$ is the collection of all finite unions of intervals of \mathcal{L} , with the usual set-theoretic operations of intersection, union and complementation. Every interval algebra is a Boolean algebra. Moreover for every countable Boolean algebra \mathcal{A} there exists an interval algebra $\mathcal{B}_{\mathcal{L}}$ isomorphic to \mathcal{A} . We write $\mathcal{L}_1 \times \mathcal{L}_2$ for the ordered sum $\sum_{l \in \mathcal{L}_2} \mathcal{L}_1$.

Lemma 3.3. *The interval Boolean algebras $\mathcal{B}_{\omega \times i}$, where i is positive integer, all have automatic presentations.*

Proof. The Boolean algebra \mathcal{B}_{ω} has an automatic presentation. Indeed, every element X of \mathcal{B}_{ω} can be represented by a string that codes the characteristic function of X . For example, the element $[1, 3) \cup [6, 10)$ can be represented by the string $0\#0110001111$ while $\mathbb{N} \setminus [3, 4)$ can be represented by the string $1\#0001$. The boolean operations under this representation are regular, hence this is an automatic presentation of \mathcal{B}_{ω} . Now, $\mathcal{B}_{\omega \times i}$ is isomorphic to the Cartesian product of i copies of \mathcal{B}_{ω} . Automatic structures are closed under the Cartesian product, and this completes the proof. \square

An **atom** in a Boolean algebra is a non-zero element a such that for every $b \leq a$ we have $a = b$ or $b = \mathbf{0}$. Assume that \mathcal{B} is an automatic Boolean algebra not isomorphic to any of the algebras $\mathcal{B}_{\omega \times i}$. Call two elements $a, b \in B$ **F-equivalent** if the element $(a \cap \bar{b}) \cup (b \cap \bar{a})$ is a union of finitely many atoms. Factorise \mathcal{B} with respect to the equivalence relation. Denote the factor algebra by \mathcal{B}/F . Due to the assumption on \mathcal{B} the algebra \mathcal{B}/F is not finite. Call x in \mathcal{B} **large** if its image in \mathcal{B}/F is not a finite union of atoms or $\mathbf{0}$. For example the element $\mathbf{1}$ is large in \mathcal{B} because \mathcal{B} is not isomorphic to $\mathcal{B}_{\omega \times i}$. Call an element x in \mathcal{B} **infinite** if there are infinitely many elements below it. Say that x **splits** y , for $x, y \in B$, if $x \cap y \neq \mathbf{0}$ and $\bar{x} \cap y \neq \mathbf{0}$. For every large element $l \in B$ there exists an element $x \in B$ that splits l such that $x \cap l$ is large and $\bar{x} \cap l$ is infinite. Also for every infinite element $i \in B$ there exists an element $x \in B$ that splits i such that either $x \cap i$ or $\bar{x} \cap i$ is infinite.

We are now ready to prove the following theorem characterising infinite automatic Boolean algebras.

Theorem 3.4. *An infinite Boolean algebra has an automatic presentation if and only if it is isomorphic to $\mathcal{B}_{\omega \times i}$ for some positive $i \in \mathbb{N}$.*

Proof. We first construct a sequence T_n of trees and elements $a_\sigma \in B$ corresponding to elements $\sigma \in T_n$ as follows. The tree T_n will be a set of binary strings closed under prefixes. Therefore it suffices to define leaves of T_n . Initially, we set $T_0 = \{\lambda\}$ and $a_\lambda = \mathbf{1}$. Assume that T_n has been constructed. By induction hypothesis we may assume that the leaves of T_n satisfy the following properties: (1) There exists at least one leaf σ such that a_σ is large in B . Call the element a_σ leading in T_n . (2) There exist n leaves $\sigma_1, \dots, \sigma_n$ such that each a_{σ_i} is an infinite element in B . Call these elements sub-leading elements. (3) The number of leaves in T_n is greater than or equal to $n \cdot (n+1)/2$. (4) For every pair of leaves x, y of T_n it holds that $x \cap y = \mathbf{0}$.

For each leaf $\sigma \in T_n$ proceed as follows:

- (1) If σ is leading then find the first length lexicographical b that splits a_σ such that both $a_\sigma \cap b$ and $a_\sigma \cap \bar{b}$ are infinite and one of them is large. Put $\sigma 0$ and $\sigma 1$ into T_{n+1} . Set $a_{\sigma 0} = a_\sigma \cap b$ and $a_{\sigma 1} = a_\sigma \cap \bar{b}$.
- (2) If σ is a sub-leading then find the first length lexicographical b that splits a_σ such that one of $a_\sigma \cap b$ or $a_\sigma \cap \bar{b}$ is infinite. Put $\sigma 0$ and $\sigma 1$ into T_{n+1} . Set $a_{\sigma 0} = a_\sigma \cap b$ and $a_{\sigma 1} = a_\sigma \cap \bar{b}$.

Thus, we have constructed the tree T_{n+1} and elements a_σ corresponding to the leaves of the tree. Note that the inductive hypothesis holds for T_{n+1} . This completes the definition of the trees.

Lemma 3.2 is now used a number of times to justify the following steps. There exists a constant c_1 such that $|a_{\sigma\epsilon}| \leq |a_\sigma| + c_1$ for all defined elements a_σ . Now for every n consider the set $X_n = \{a_\sigma \mid \sigma \text{ is a leaf of } T_n\}$. There exists a constant c_2 such that for all $x \in X_n$ we have $|x| \leq c_2 \cdot n$. Therefore $X_n \subset \Sigma^{c_2 \cdot n}$ and the number of leaves in T_n is greater than or equal to $n \cdot (n+1)/2$. Now for every pair of elements a, b in X_n we have $a \cap b = \mathbf{0}$. Therefore the number of elements of the Boolean algebra generated by the elements in X_n is $2^{|X_n|}$. Now let $Y = \{b_1, \dots, b_k\} \subset X_n$. Consider the element $\cup Y = b_1 \cup \dots \cup b_k$. By Lemma 3.2 applied to the binary operation \cup there exists a constant c_3 such that $|\cup Y| \leq c_3 \cdot n$. This gives us a contradiction because the number of elements generated by elements of X_n clearly exceeds the cardinality of $\Sigma^{O(n)}$. \square

Corollary 3.5. *It is decidable whether two automatic Boolean algebras are isomorphic.*

Proof. Every automatic Boolean algebra is isomorphic to the Cartesian product of i copies of B_ω , the Boolean algebra of finite and co-finite subsets of \mathbb{N} . This i can be obtained effectively: Given an FA-presentation of a structure A in the signature of Boolean algebras, one can decide if A is a Boolean algebra, and if so compute the largest i such that A models

“there are i disjoint elements each with infinitely many atoms below.”

Thus the isomorphism problem is decidable. \square

3.2. Commutative monoids and Abelian groups. Note that for groups and monoids the term ‘automatic’ is used in a different way [6]. So to avoid confusion we say such a structure is ‘FA presented’ instead of saying it is ‘automatic’, and it is ‘FA presentable’ instead of ‘automatically presentable’.

We prove that (\mathbb{Q}^+, \times) , or equivalently, the free Abelian group of rank ω , is not FA presentable.

Theorem 3.6. *Let (M, \times) be a monoid containing (\mathbb{N}, \times) as a submonoid. Then (M, \times) is not FA presentable.*

Proof. Assume for a contradiction that an FA presentation of M is given. Let a_0, a_1, \dots be the prime numbers, viewed as elements of M , and listed in length-lexicographical order (with respect to this presentation of M). Let r_n be such that a_0, \dots, a_{r_n-1} are the primes of length at most n . Let

$$F_n = \{\prod_{i: 0 \leq i < r_n} a_i^{\beta_i} : 0 \leq \beta_i < 2^n\}.$$

By Lemma 3.2, each term $a_i^{\beta_i}$ has length at most $n + k \log \beta_i \leq n(1 + k)$. Again by the lemma, each product has length at most $n(1 + k) + k \log r_n$. Since all the products are distinct,

$$2^{nr_n} \leq |F_n| \leq |\Sigma|^{(1+k)n + k \log r_n}.$$

Thus $nr_n \leq \log |\Sigma|[(1 + k)n + k \log r_n]$ and $r_n \in O(\log r_n)/n$, a contradiction because r_n goes to infinity. \square

In [15] a stronger form is proved: if $(\mathbb{N}, +)^r$ is a submonoid of M , then $r \leq \log |\Sigma|(k + 1)$, where Σ is the alphabet and k the number of states needed to recognize the graph of the operation of M .

Corollary 3.7. (\mathbb{Q}^+, \times) is not FA presentable. \square

For a prime p , let $\mathbb{Z}[1/p]$ be the additive group of rationals of the form z/p^m , z an integer, $m \in \mathbb{N}$, and let \mathbb{Z}_{p^∞} be the Prüfer group $\mathbb{Z}[1/p]/\mathbb{Z}$. Using representations to base p , it is easy to give FA presentations of these groups. Hence finite direct sums of those groups are also FA presentable. The proof of the following uses similar methods to the ones of Theorem 3.6.

Theorem 3.8. *Let $A^{(\omega)}$ denote the direct sum of infinitely many copies of the group A . The infinite direct sums $\mathbb{Z}[1/p]^{(\omega)}$ and $\mathbb{Z}_{p^\infty}^{(\omega)}$ are not FA presentable.*

Proof. The proof is similar to the proof of Theorem 3.6. Suppose there is an FA presentation, and the i -th copy of the group in question is generated by all elements of the form a_i/p^m , $i, m \in \mathbb{N}$, where the elements a_0, a_1, \dots are listed in length-lexicographical order. Define r_n as before, and consider sums $\sum_{i < r_n} \beta_i a_i/p^n$, where $0 \leq \beta_i < p^n$. By Proposition 3.1, the definable operation $x \mapsto x/p$ increases the length by at most a constant. So, using Lemma 3.2 each term $\beta_i a_i/p^n$ has length at most $n + cn + kn \log p$ for appropriate $c \in \mathbb{N}$. Thus each sum has length at most $c'n + k \log r_n$. As there are p^{nr_n} distinct sums, this yields a contradiction as before. \square

3.3. Integral domains. In our next result we prove that no infinite integral domain is FA presentable. The following definition and lemma will be used in the next section as well.

Definition 3.1. Suppose D is a structure over alphabet Σ . Write $D^{\leq n}$ for $D \cap \Sigma^{\leq n}$; that is the elements of D of length at most n . Write $P_n(D)$ for $\{x \in \Sigma^n \mid \exists z \in \Sigma^* \wedge xz \in D\}$, namely all prefixes of length n of all words in the domain.

Lemma 3.9. *If $D \subset \Sigma^*$ is a regular language then*

- (1) $|P_n(D)| \in O(|D^{\leq n}|)$ and

(2) $|D^{\leq n+k}| \in \Theta(|D^{\leq n}|)$ for every constant $k \in \mathbb{N}$.

Proof. Suppose the automaton recognising D has c states. Then for $x \in P_n(D)$ there exists $z \in \Sigma^*$ with $|z| \leq c$ such that $xz \in D$ (\dagger). If $n \geq c$ then $|P_n(D)| \leq |\Sigma|^c \times |P_{n-c}(D)|$ since the map associating $x \in P_n(D)$ with the word consisting of the first $n - c$ letters of x , is $|\Sigma|^c$ -to-one. But by using (\dagger) we see that $|P_{n-c}(D)| \leq |D^{\leq n}|$. So $|P_n(D)| \leq |\Sigma|^c \times |D^{\leq n}|$ as required for the first part.

Fix $k \in \mathbb{N}$. The mapping associating $x \in D^{\leq n+k}$ to the prefix of x of length n is $|\Sigma|^k$ -to-one. Hence

$$|D^{\leq n+k}| \leq |\Sigma|^k \times |P_n(D)| \leq |\Sigma|^k \times |\Sigma|^c \times |D^{\leq n}|.$$

Since $D^{\leq n} \subset D^{\leq n+k}$ one has that $|D^{\leq n}| \leq |D^{\leq n+k}|$ and $|D^{\leq n+k}| \in O(|D^{\leq n}|)$. This completes the proof of the lemma. \square

Recall that an integral domain $(D, +, \cdot, 0, 1)$ is a commutative ring with identity such that $x \cdot y = 0$ only if $x = 0$ or $y = 0$. For example every field is an integral domain.

Theorem 3.10. *No infinite integral domain is FA presentable.*

Proof. Suppose that $(D, +, \cdot, 0, 1)$ is an infinite automatic integral domain. For each $n \in \mathbb{N}$ recall that $D^{\leq n} = \{u \in D \mid |u| \leq n\}$. We claim that there exists an x in D such that for all $a, b, a', b' \in D^{\leq n}$ the condition $a \cdot x + b = a' \cdot x + b'$ implies that $a = a'$ and $b = b'$. We say such x **separates** $D^{\leq n}$. Indeed, assume that such an x does not exist. Then for each $x \in D$ there exist $a, b, a', b' \in D^{\leq n}$ such that $a \cdot x + b = a' \cdot x + b'$ but $(a, b) \neq (a', b')$. Hence, since $D^{\leq n}$ is finite, there exist $a, b, a', b' \in D^{\leq n}$ such that $a \cdot x + b = a' \cdot x + b'$ but $(a, b) \neq (a', b')$ for infinitely many x . Thus, for infinitely many x we have $(a - a') \cdot x = b' - b$. But $a \neq a'$ for otherwise also $b = b'$, contrary to assumption. Also there exist distinct x_1 and x_2 such that $(a - a') \cdot x_1 = (a - a') \cdot x_2$. Since D is an integral domain we conclude that $x_1 = x_2$ which is a contradiction.

For each $D^{\leq n}$ we can select the length-lexicographically first x_n separating $D^{\leq n}$. Now the set $E_n = \{y \mid \exists a, b \in D^{\leq n} [y = ax_n + b]\}$ has at least $|D^{\leq n}|^2$ many elements. However by Proposition 3.1 there exists a constant C such that $E_n \subset D^{\leq n+C}$, and by Lemma 3.9 the number of elements in $D^{\leq n+C}$ is in $O(|D^{\leq n}|)$. Thus, we have a contradiction. The theorem is proved. \square

Corollary 3.11. *No infinite field is FA presentable.* \square

4. NON-AUTOMATICITY OF SOME FRAÏSSÉ LIMITS

We briefly recall the definition of Fraïssé limit; see for instance [10]. In this section we restrict consideration to relational structures of finite signature, although the definition can be extended to signatures with function symbols. Let K be a class of finite structures closed under isomorphism. We say that K has the **hereditary property (HP)** if for $\mathcal{A} \in K$ every substructure of \mathcal{A} is also in K . We say that K has the **joint embedding property (JEP)** if for all $\mathcal{A}, \mathcal{B} \in K$ there exists $\mathcal{C} \in K$ such that \mathcal{A} and \mathcal{B} are both embeddable into \mathcal{C} . We say that K is the **age** of a structure \mathcal{D} if K coincides with the class of all finite substructures of \mathcal{D} . It is clear that the age of a structure has HP and JEP. In fact, it is not hard to prove that a class K has HP and JEP if and only if K coincides with the class of all finite substructure of some structure. However, note that non-isomorphic structures

may have the same age. The following property guarantees that a class K with HP and JEP defines a unique structure up to isomorphism:

Definition 4.1. A class K of finite structures has **amalgamation property (AP)** if for $\mathcal{A}, \mathcal{B}, \mathcal{C} \in K$ with embeddings $e : \mathcal{A} \rightarrow \mathcal{B}$ and $f : \mathcal{A} \rightarrow \mathcal{C}$ there are $\mathcal{D} \in K$ and embeddings $g : \mathcal{B} \rightarrow \mathcal{D}$ and $h : \mathcal{C} \rightarrow \mathcal{D}$ such that $ge = hf$.

Below we cite a classical result in model theory due to Fraïssé . For this we mention that a structure \mathcal{A} is called **ultra-homogeneous** if every isomorphism between finite substructures of \mathcal{A} can be extended to an automorphism of \mathcal{A} . For the proof the reader may consult [10, Theorem 7.1.2].

Theorem 4.1. *Let K be a nonempty class of finite structures which has HP, JEP, and AP. Then there exists a up to isomorphism unique countable structure \mathcal{A} , called the **Fraïssé limit** of K , such that \mathcal{A} is ultra-homogeneous and K is the age of \mathcal{A} . \square*

Now we can apply the theorem to obtain several examples of structures. In each of these examples the classes K have HP, JEP, and AP.

Example 4.2. Let K be the class of all finite linear orders. The Fraïssé limit of K is isomorphic to the ordering of rationals.

Example 4.3. Let K be the class of all finite linear orders with one unary predicate. The Fraïssé limit of K is isomorphic $(Q; \leq, U)$, where \leq is the linear order of rationals Q , and U is a dense and co-dense subset of Q .

Example 4.4. Let K be the class of all finite graphs. The Fraïssé limit of K is known as the **random graph**. The following is an algebraic property that gives a characterisation (of the isomorphism type) of the random graph $\mathcal{R} = (V, E)$ (see [10]). For every disjoint partition X_1, X_2 of every finite set Y of vertices there exists a vertex y such that for all $x_1 \in X_1$ and $x_2 \in X_2$ we have $(y, x_1) \in E$ and $(y, x_2) \notin E$. An explicit description of a random graph is the following. The set V of vertices is \mathbb{N} and $(n, m) \in E$ if in the binary representation of n , the term 2^m has coefficient 1.

Example 4.5. Let K be the class of all finite structures of a given signature L . The Fraïssé limit of K is known as the **random L -structure**.

Example 4.6. Denote by K_p the complete graph (every pair of vertices are connected by an edge) on p vertices. For $p \geq 3$, consider the class of all finite graphs which do not contain K_p as a subgraph. The Fraïssé limit of K is known as the **random K_p -free graph**. It has the following property. For every finite K_{p-1} -free subgraph, say with domain Y , and every disjoint partition X_1, X_2 of Y , there exists a vertex x that is edge connected to every vertex in X_1 and no vertex in X_2 .

Recall that an anti-chain in a partial order is a set of pairwise incomparable elements. A chain in a partial order is a set in which every pair of elements are comparable.

Example 4.7. Let K be the class of all finite partially ordered set. The Fraïssé limit \mathcal{U} of K is known as the **random partial order**. The following is an algebraic characterisation (of the isomorphism type) of the random partial order $\mathcal{U} = (U, \leq)$ (see [10]).

- (1) If Z is a *finite* anti-chain of \mathcal{U} and X and Y partition Z then there exists an element $z \in U$ such that for every $x \in X$, $z > x$ and for every $y \in Y$, element z is not comparable with y .

- (2) If Z is a *finite* chain of \mathcal{U} with least element x and largest element y then there exists an element $z \in U$ such that $z > x$ and $z < y$ and z is not comparable with every $v \in X \setminus \{x, y\}$.

Example 4.8. Let K be the class of all finite Boolean algebras. The Fraïssé limit of K is isomorphic to the atomless Boolean algebra. This is the Boolean algebra that satisfies the following property. For every non-zero element x there exists a nonzero y strictly below x (that is $y < x$). By Theorem 3.4 this Fraïssé limit has no automatic presentation.

Example 4.9. Let K be the class of all finite Abelian p -groups. The Fraïssé limit of K is isomorphic to $G = (\mathbb{Z}_{p^\infty})^\omega$.

Note that G has no FA-presentation by Theorem 3.8.

Proof. The class K has HP, JEP and AP, so the Fraïssé limit exists. To show this Fraïssé limit is isomorphic to G , by [10, Lemma 7.1.4] it suffices to show that the age of G is K (clear) and that G is weakly homogeneous. To do so, suppose $A \subseteq B$ are in K . We have to show that each embedding of A into G extends to an embedding of B . We may assume that $|B : A| = p$. Since B is a direct product of cyclic groups whose order is a power of p , either $B = A \times \mathbb{Z}_p$ or there is $x \in A$ such that x is not divisible by p in A , and $py = x$ for some $y \in B$. In either case we can extend the embedding. \square

Below are examples of Fraïssé limits that have automatic presentations.

Example 4.10. The linear order of rational numbers has an automatic presentation. In fact it is straightforward to check that $(\{0, 1\}^* \cdot 1, \preceq_{lex})$ is an automatic presentation of (Q, \leq) .

Example 4.11. Let $U = \{u \mid u \in \{0, 1\}^* \cdot 1, |u| \text{ is even}\}$. The structure $(\{0, 1\}^* \cdot 1; \preceq_{lex}, U)$ is the Fraïssé limit for the class K of all finite linear orders with one unary predicate.

We now provide some methods for proving non-automaticity of structures. These methods are then applied to prove that some Fraïssé limits do not have automatic presentations.

Let \mathcal{A} be an automatic structure over the alphabet Σ . Recall $A^{\leq n} = \{v \in A \mid |v| \leq n\}$. Let $\Phi(x, y)$ be a two variable formula in the language of this structure. We do not exclude that $\Phi(x, y)$ has a finite number of parameters from the domain of the structure. Now for each $y \in A$ and $n \in \mathbb{N}$ we define the following function $c_{n,y}^\Phi : A^{\leq n} \rightarrow \{0, 1\}$:

$$c_{n,y}^\Phi(x) = \begin{cases} 1 & \text{if } \mathcal{A} \models \Phi(x, y); \\ 0 & \text{if } \mathcal{A} \models \neg\Phi(x, y). \end{cases}$$

We may drop the superscript Φ if there is no danger of ambiguity. In the next theorem we count the number of functions $c_{n,y}^\Phi$ using the fact that \mathcal{A} is an automatic structure. We will use this as a criterion for proving that a given structure is not automatically presentable.

Theorem 4.2. *Let \mathcal{A} be an automatic structure and $\Phi(x, y)$ a first order formula (possibly with parameters) over the language of \mathcal{A} . Then the number of functions $c_{n,y}^\Phi$ is in $O(|A^{\leq n}|)$.*

Proof. It is sufficient to prove that there is a constant k so that the number of functions of the form $c_{n,y}$ with $y \in A \cap \Sigma^{>n}$ is at most $k(|A^{\leq n}|)$; this is because the y 's in $A^{\leq n}$ can supply at most $|A^{\leq n}|$ many additional functions $c_{n,y}$.

Let (Q, ι, ρ, F) be a deterministic automaton recognising the relation $\otimes\Phi = \{\otimes(x, y) \mid \mathcal{A} \models \Phi(x, y)\}$. Fix $n \in \mathbb{N}$. We will associate with each $c_{n,y}$, where $y \in A \cap \Sigma^{>n}$, two

pieces of information; namely, a function $J_y : A^{\leq n} \rightarrow Q$ and a set $Q_y \subset Q$ as follows. Let $\otimes(x, y) = \sigma_1\sigma_2 \cdots \sigma_k$, where $x \in A^{\leq n}$ and $\sigma_i \in (\Sigma_\circ)^2$. Then define $J_y(x) := \rho(\iota, \sigma_1 \cdots \sigma_n)$. Define $Q_y \subset Q$ as those states $s \in Q$ such that $\rho(s, \sigma_{n+1} \cdots \sigma_k) \in F$. Note that $\sigma_{n+1} \cdots \sigma_k = \otimes(\lambda, z)$, for some $z \in \Sigma^*$, and is independent of x .

We claim that if $(J_y, Q_y) = (J_{y'}, Q_{y'})$ then $c_{n,y} = c_{n,y'}$. So suppose that $(J_y, Q_y) = (J_{y'}, Q_{y'})$, and let $x \in A^{\leq n}$ be given. Say $\otimes(x, y) = \sigma_1 \cdots \sigma_k$, and $\otimes(x, y') = \delta_1 \cdots \delta_l$. Then

$$\begin{aligned} \mathcal{A} \models \phi(x, y) &\iff \rho(\iota, \otimes(x, y)) \in F \\ &\iff \rho(J_y(x), \sigma_{n+1} \cdots \sigma_k) \in F \\ &\iff J_y(x) \in Q_y \\ &\iff J_{y'}(x) \in Q_{y'} \\ &\iff \rho(J_{y'}(x), \delta_{n+1} \cdots \delta_l) \in Q_{y'} \\ &\iff \mathcal{A} \models \phi(x, y'). \end{aligned}$$

Then $c_{n,y} = c_{n,y'}$ as claimed. Thus the number of functions of the form $c_{n,y}$ ($y \in A \cap \Sigma^{>n}$) is at most the number of distinct pairs of the type (J_y, Q_y) . Now the number of Q_y is at most $2^{|Q|}$, so we concentrate on bounding the number of distinct functions J_y by $O(|A^{\leq n}|)$.

Note that J_y depends only on the first n letters of y in the sense that for $|v| \geq n$ and $w, w' \in \Sigma^*$, $J_{vw} = J_{vw'}$. Now every $y \in A \cap \Sigma^{>n}$ can be decomposed as $y = vw$ with $|v| = n$. But the first part of Lemma 3.9 says that there are $O(|A^{\leq n}|)$ many such y , and so the result follows. \square

We give several applications of this theorem. First we apply the theorem to random graphs.

Corollary 4.3. (independently [7]) *The random graph has no automatic presentation.*

Proof. Let (A, E) be an automatic presentation of the random graph and let $\Phi(x, y)$ be $E(x, y)$. For every partition X_1, X_2 of the set $A^{\leq n}$ of vertices there exists a vertex y such that for all $x_1 \in X_1$ and $x_2 \in X_2$ it holds that $(x_1, y) \in E$ and $(x_2, y) \notin E$. Hence, for every n , we have $2^{|A^{\leq n}|}$ number of functions of type $c_{n,y}$, contradicting Theorem 4.2. Hence the random graph has no automatic presentation. \square

Corollary 4.4. *Let \mathcal{A} be a random structure of a signature L , where L contains at least one non-unary symbol. Then \mathcal{A} does not have an automatic presentation.*

Proof. Let R be a non-unary predicate of L of arity k . Consider the following formula $E(x, y) = R(x, y, a_1, \dots, a_{k-2})$, where a_1, \dots, a_{k-2} are fixed constants from the domain of \mathcal{A} . It is not hard to prove that (A, E) is isomorphic to the random graph. But if \mathcal{A} is automatically presentable then so is the random graph (A, E) , hence contradicting the previous corollary. \square

The next goal is to show that the random K_p -free graph has no automatic presentation. For this one needs to have a finer analysis than the one for the random graph. Let $\mathcal{F} = (V, E)$ be a finite graph. For a vertex v , write $E(v)$ for the set of vertices adjacent to v . The *degree* of a vertex is the cardinality of $E(v)$. Write $\Delta(\mathcal{F})$ for the maximum degree over all the vertices of \mathcal{F} . Call a subgraph \mathcal{G} with no edges an *independent* graph. Let $\alpha(\mathcal{F})$ be the number of vertices of a largest independent subgraph of \mathcal{F} .

K_p denotes the complete graph on p vertices; that is, there is an edge between every pair of vertices. A graph is called K_p -*free* if it has no subgraph isomorphic to K_p .

Lemma 4.5. *For every $p \geq 3$, there is a polynomial $Q_p(x)$ of degree $p - 1$ so that if \mathcal{F} is a finite K_p -free graph then $Q_p(\alpha(\mathcal{F})) \geq |F|$.*

Proof. We first prove that for every finite graph \mathcal{F} ,

$$\alpha(\mathcal{F}) \geq |F|/(\Delta(\mathcal{F}) + 1). \quad (4.1)$$

Let \mathcal{G} be an independent subgraph of \mathcal{F} with a maximal number of vertices. That is, $\alpha(\mathcal{F}) = |\mathcal{G}|$. For every $d \in G$ let $N(d) = E(d) \cup \{d\}$, where $E(d)$ is the set of vertices in \mathcal{F} adjacent to d . Then since \mathcal{G} is maximal, for every $x \in F$ there is some (not necessarily unique) $d \in G$ such that $x \in N(d)$. Hence $\mathcal{F} = \cup_{d \in G} N(d)$. But $|N(d)|$ equals the degree (in \mathcal{F}) of d plus one, and so the largest cardinality amongst the $N(d)$'s is at most $\Delta(\mathcal{F}) + 1$. Hence $|F| \leq |G| \times (\Delta(\mathcal{F}) + 1)$ as required.

The lemma is proved by induction on p . We will show that $Q_p(x) = \sum_{i=1}^{p-1} x^i$. For the case $p = 3$ note that for every vertex v , the subgraph on domain $E(v)$ is independent. For otherwise if $x, y \in E(v)$ were joined by an edge then the subgraph of \mathcal{F} on $\{x, y, v\}$ is K_3 . In particular, $\alpha(\mathcal{F}) \geq \Delta(\mathcal{F})$. Combining this with Inequality (4.1), we get $\alpha(\mathcal{F})[\alpha(\mathcal{F})+1] \geq |F|$ as required.

For the inductive step, let \mathcal{F} be a K_p -free graph with $p > 3$. For every vertex v , the set $E(v)$ is K_{p-1} -free for otherwise the subgraph of \mathcal{F} on $E(v) \cup \{v\}$ has a copy of K_p . Applying the induction hypothesis to $E(v)$ such that $|E(v)| = \Delta(\mathcal{F})$, we get that $E(v)$ must have an independent set X so that $Q_{p-1}(|X|) \geq |E(v)|$. But X is also independent in \mathcal{F} so $Q_{p-1}(\alpha(\mathcal{F})) \geq \Delta(\mathcal{F})$. Combining this with Inequality 4.1, we get that $\alpha(\mathcal{F})[Q_{p-1}(\alpha(\mathcal{F})) + 1] \geq |F|$. Hence $Q_p(\alpha(\mathcal{F})) \geq |F|$ as required. \square

Corollary 4.6. *For $p \geq 3$, the random K_p -free graph is not automatically presentable.*

Proof. Fix $p \geq 3$ and let (D, E) be a copy of the random K_p -free graph. Then for every K_{p-1} -free subset $K \subset D^{\leq n}$ there exists an $x \in D$ that is connected to every vertex in K and none in $D^{\leq n} \setminus K$. So let \mathcal{G}_n be an independent subgraph of $D^{\leq n}$ so that $Q_p(|\mathcal{G}_n|) \geq |D^{\leq n}|$ as in the lemma. So letting $\Phi(x, y)$ be $E(x, y)$, for a fixed n the number of functions $c_{n,y}$ as y varies over D is at least $2^{|G_n|}$ which is not linear in $|D^{\leq n}|$. Hence by Theorem 4.2 the random K_p graph \mathcal{D} is not automatically presentable. \square

As the fourth application we prove that the random partial order \mathcal{U} does not have an automatic presentation. For the proof we need the following combinatorial result that connects the size of a finite partial order (B, \leq) with the cardinalities of its chains and anti-chains.

Lemma 4.7 (Dilworth). *Let (B, \leq) be a finite partial order of cardinality n . Let a be the size of largest anti-chain in (B, \leq) and let c be the size of the largest chain in (B, \leq) . Then $n \leq ac$.*

Proof. For $1 \leq i \leq c$ define X_i as the set of all elements x such that the size of the largest chain in the subpartial order $(\uparrow x) = \{y \in B \mid x \preceq y\}$ is i . Then the X_i 's partition B . Moreover if $a \prec b$ and $b \in X_i$ then the size of the largest chain in $(\uparrow a)$ is $> i$. Hence each X_i is an anti-chain. Thus B can be partitioned into exactly c many anti-chains. If a is the size of the largest anti-chain in B then $n \leq ac$ as required. \square

Corollary 4.8. *The random partial order $\mathcal{U} = (U, \leq)$ has no automatic presentation.*

Proof. Recall that the random partial order has the following property

- (1) If Z is a *finite* anti-chain of \mathcal{U} , and X and Y partition Z , then there exists an element $z \in U$ such that $z > x$ for every $x \in X$, and z is not comparable with y for every $y \in Y$.
- (2) If Z is a *finite* chain of \mathcal{U} with least element x and largest element y , then there exists an element $z \in U$ such that $z > x$ and $z < y$ and z is not comparable with every $v \in X \setminus \{x, y\}$.

Assume that \mathcal{U} has an automatic presentation (A, \leq) . The formula $\Phi(x, y)$ is $x \leq y \vee y \leq x$. Now let us take $A^{\leq n}$.

Let Z be an anti-chain of $A^{\leq n}$. Consider a subset X of Z . There exists an element $y \in U$ such that for every $x \in X$, $y > x$ and for every $x' \notin Z \setminus X$, element y is not comparable with x' . From this we conclude that

$$(\star) \quad \#(\text{functions of type } c_{n,y}) \geq 2^{|Z|}.$$

Let Z be a chain of $A^{\leq n}$ with least element v and largest element w . Then there exists an element $y \in U$ such that $y > v$ and $y < w$ and y is not comparable with x for every $x \in Z$. From this we conclude that

$$(\star\star) \quad \#(\text{functions of type } c_{n,y}) \geq \binom{|Z|}{2}.$$

Let X be the largest anti-chain and Y be the largest chain of $A^{\leq n}$ with cardinalities a and c , respectively. Using Dilworth Lemma, substituting a for $|Z|$ in (\star) , and c for $|Z|$ in $(\star\star)$, one can, with a little algebra, derive a contradiction to the bound in the statement of Theorem 4.2. \square

5. THE ISOMORPHISM PROBLEM

The results in the previous sections give us hope that one can characterise automatic structures for certain classes of structures, e.g. Boolean algebras. However, in this section we prove that the isomorphism problem is Σ_1^1 -complete, thus showing that the problem is as hard as possible when considering the class of *all* automatic structures. Then **complexity of the isomorphism problem** for automatic structures consists in establishing the complexity of the set $\{(\mathcal{A}, \mathcal{B}) \mid \mathcal{A} \text{ and } \mathcal{B} \text{ are automatic structures and } \mathcal{A} \text{ is isomorphic to } \mathcal{B}\}$.

Let \mathcal{M} be a Turing machine over input alphabet Σ . Its configuration graph $C(\mathcal{M})$ is the set of all configurations of \mathcal{M} , with an edge from c to d if T can move from c to d in a single transition. The following is an easy lemma:

Lemma 5.1. *For every Turing machine T the configuration graph $C(T)$ is automatic. Further, the set of all vertices with outdegree (indegree) 0 is FA-recognisable.* \square

Definition 5.1. A Turing machine \mathcal{R} is **reversible** if every vertex in $C(\mathcal{R})$ has both indegree and outdegree at most one.

Now let \mathcal{R} be a reversible Turing machine. Consider its configuration space $C(\mathcal{R})$. The machine \mathcal{R} can be modified so that it only halts in an accepting state; so, instead of halting in a rejecting state, it loops forever. Let x be a configuration of \mathcal{R} . Consider the sequence: $x = x_0, x_1, x_2, \dots$ such that $(x_i, x_{i+1}) \in E$, where E is the edge relation of the configuration space. Call this sequence the **chain defined by x** . We say that x is the **base** of chain X . If x does not have a predecessor then the chain defined by x is maximal. Since \mathcal{R} is reversible, the configuration space $C(\mathcal{R})$ is a disjoint union of maximal chains such that each chain is either finite, or isomorphic to (\mathbb{N}, S) , or isomorphic to (\mathbb{Z}, S) , where $(x, y) \in S$ iff $y = x + 1$. It is known that every Turing machine can be converted into an equivalent reversible Turing machine (see for example [3]). Our next lemma states this fact and provides some additional structural information about the configuration space of reversible Turing machines:

Lemma 5.2. *A deterministic Turing machine can be converted into an equivalent reversible Turing machine \mathcal{R} such that every maximal chain in $C(\mathcal{R})$ is either finite or isomorphic to (\mathbb{N}, S) .*

Denote by \mathbb{N}^* the set of all finite strings from \mathbb{N} . A set $T \subset \mathbb{N}^*$ is a *special tree* if T is closed downward, namely $xy \in T$ implies $x \in T$, for $x, y \in \mathbb{N}^*$. We view these special trees as structures of the signature E , where $E(x, y)$ if and only if $y = xz$ for some $z \in \mathbb{N}$. Thus, for every $x \in \mathbb{N}^*$ the set $\{y \mid E(x, y)\}$ can be thought as the set of all *immediate successors* of x .

A special tree T is *recursively enumerable* if the set T is the domain of the function computed by a Turing machine. We will use the following fact from computable model theory [9, Thm 3.2].

Lemma 5.3. *The isomorphism problem for recursively enumerable special trees is Σ_1^1 -complete. In fact, the trees can be chosen to be subtrees of $\{2n : n \in \mathbb{N}\}^*$.*

It is clear from the proof in [9] that the trees obtained are special (namely, subtrees of \mathbb{N}^*). By a mere change of notation one obtains subtrees of $\{2n : n \in \mathbb{N}\}^*$.

Lemma 5.4. *The special tree \mathbb{N}^* has an automatic presentation \mathcal{A}_1 .*

Proof. Consider the prefix relation \leq_p on the set of all binary strings. The tree \mathbb{N}^* is isomorphic to the automatic successor tree $\mathcal{A}_1 = (\{0, 1\}^*1 \cup \{\lambda\}; E_1)$, where E_1 is the set of all pairs (x, y) such that y is the immediate \leq_p -successor of x . The isomorphism is established via the computable mapping sending $n_1 \dots n_k$ to $0^{n_1}1 \dots 0^{n_k}1$ and the root to λ . \square

Define $\mathcal{S} = \{0, 1\}^*1 \cup \{\lambda\}$. From now on we make the following conventions:

- (1) All special trees we consider will be viewed as subsets of (\mathcal{S}, \leq_p) .
- (2) The set of inputs for all Turing machines considered are strings from \mathcal{S} . Each $w \in \mathcal{S}$ is identified with the initial configuration starting from w .
- (3) All Turing machines considered are reversible.
- (4) The domains of the functions computed by the Turing machines are assumed to be downward closed. Thus, these domains form recursively enumerable special trees.

Our goal is to transform every recursively enumerable tree $T \subset \mathcal{S}$ into an automatic structure $\mathcal{A}_{\mathcal{R}}$, where \mathcal{R} is the reversible machine with domain T . The idea is to attach computations of \mathcal{R} to the initial configurations in \mathcal{S} . For all recursively enumerable trees T_1

and T_2 , T_1 and T_2 are isomorphic if and only if the automatic structures $\mathcal{A}_{\mathcal{R}_1}$ and $\mathcal{A}_{\mathcal{R}_2}$ are isomorphic. To ensure this we also have to attach infinitely many chains of each finite length to the initial configurations, for in that case the length of a halting computation does not make a difference.

Let \mathcal{R} be a reversible Turing machine. A **chain** is an initial segment of $(\mathbb{N}, \mathcal{S})$. Let \mathcal{J} be the graph consisting of infinitely many finite chains of every finite length. We denote by J the set of vertices of \mathcal{J} , and the bases of the chains in \mathcal{J} by b_1, b_2, \dots . The following is not hard to prove:

Lemma 5.5. *The graph \mathcal{J} has an automatic presentation.*

To construct $\mathcal{A}_{\mathcal{R}}$, with every $w \in A_1$ associate a graph \mathcal{J}_w defined as follows. The vertex set of \mathcal{J}_w consists of all $\{(w, j) \mid j \in J\}$ and edges between (w, j) and (w, j') if and only if (j, j') is an edge in \mathcal{J} . Put connecting edges from w into each (w, b_i) . Let \mathcal{A}_2 be the graph consisting of A_1 and every \mathcal{J}_w and the connecting edges. Clearly, the graph \mathcal{A}_2 is automatic.

To the set A_2 add all the configurations of the Turing machine \mathcal{R} and all the edges of the configuration space of \mathcal{R} . Note that each $w \in \mathcal{S}$, which is an initial configuration of \mathcal{R} is by Convention 2 already in A_2 . In addition, add a disjoint automatic copy of the graph \mathcal{J} , and an automatic copy of infinitely many infinite chains. Call all chains added at this stage **junk chains**. The resulting graph is denoted by $\mathcal{A}_{\mathcal{R}}$.

The proof of the following is straightforward and is omitted. □

Lemma 5.6. *The graph $\mathcal{A}_{\mathcal{R}}$ is automatic.* □

We summarise the structure of the graph $\mathcal{A}_{\mathcal{R}}$. Firstly, in $\mathcal{A}_{\mathcal{R}}$ there are infinitely many junk chains of every (finite and infinite) length. An **isolated chain** of $\mathcal{A}_{\mathcal{R}}$ is one in which every vertex, except the base, has exactly one successor in $\mathcal{A}_{\mathcal{R}}$.

With each element w in A_1 there is an associated structure \mathcal{J}_w , which consists of infinitely many isolated chains of every finite length, and no infinite isolated chain. Finally with every initial configuration w (which is an element of $\mathcal{A}_{\mathcal{R}}$ and belongs to the infinitely branching tree A_1) there is an isolated chain with base w that is the computation path from the configuration space of Turing machine T . These observations prove the following.

Lemma 5.7. *Let w be an initial configuration of $\mathcal{A}_{\mathcal{R}}$. Then w is the base of infinitely many isolated chains of every finite length. Also the Turing machine T halts on w if and only if there is no infinite isolated chain with base w . In case T does not halt on w there is exactly one infinite isolated chain with base w .* □

Lemma 5.8. *Let T_1 and T_2 be computable trees which are domains of reversible Turing machines \mathcal{R}_1 and \mathcal{R}_2 respectively. Then the trees T_1 and T_2 are isomorphic if and only if the automatic graphs $\mathcal{A}_{\mathcal{R}_1}$ and $\mathcal{A}_{\mathcal{R}_2}$ are isomorphic.*

Proof. First note that the domain T of \mathcal{R} consists of all w in $\mathcal{A}_{\mathcal{R}}$ that are initial configurations and are not the base of an infinite isolated chain. Hence T may be thought of as a subgraph of $\mathcal{A}_{\mathcal{R}}$. Now suppose $\mathcal{A}_{\mathcal{R}_1}$ is isomorphic to $\mathcal{A}_{\mathcal{R}_2}$ via the map ϕ . Note that w is the base of an infinite isolated chain if and only if $\phi(w)$ is the base of an infinite isolated chain. Hence ϕ is an isomorphism between the trees T_1 and T_2 viewed as subgraphs of $\mathcal{A}_{\mathcal{R}_1}$ and $\mathcal{A}_{\mathcal{R}_2}$ respectively.

Conversely assume T_1 and T_2 are isomorphic via ψ (as before we may assume that T_i is a subgraph of $\mathcal{A}_{\mathcal{R}_i}$). Consider the sets I_1 and I_2 of all junk chains in $\mathcal{A}_{\mathcal{R}_1}$ and $\mathcal{A}_{\mathcal{R}_2}$. There

is an isomorphism between I_1 and I_2 , since both consist of infinitely many chains of every length (finite and infinite), independently of the chains of the configuration graphs that are *not* defined by initial configurations. Therefore it suffices to establish an isomorphism extending ψ on the rest of the structure.

If $w \in T_1$ then there is an isomorphism between all the isolated chains with base w and all the isolated chains with base $\psi(w)$. Indeed each is the base of infinitely many isolated chains of every finite length and no isolated chain of infinite length, independently of the (possibly different) lengths of the finite computations of R_1 on w and R_2 on $\psi(w)$. Hence extend ψ from the chains defined by elements of T_1 to the chains defined by elements of T_2 .

Suppose $w \in T_1$ and consider the set S_1 of immediate successors of w that are initial configurations but not in T_1 . Similarly write S_2 for the set of immediate successors of $\psi(w)$ that are initial configurations but not in T_2 . By the extra condition in Lemma 5.3 that the trees be subtrees of $\{2n : n \in \mathbb{N}\}^*$, both S_1 and S_2 are infinite. So we may extend ψ to those immediate successors by adding a bijection between S_1 and S_2 .

Finally, for any initial configurations $v_1 \in \mathcal{S}-T_1$ and $v_2 \in \mathcal{S}-T_2$, there is an isomorphism between the nodes in \mathcal{S} extending v_1 and the ones extending v_2 . This isomorphism extends to the attached chains (as there is one infinite isolated chain and infinitely many isolated chains of each finite length). So we may extend ψ to an isomorphism of $\mathcal{A}_{\mathcal{R}_1}$ and $\mathcal{A}_{\mathcal{R}_2}$. \square

Hence we have reduced the isomorphism problem for recursively enumerable trees to the isomorphism problem for automatic graphs. The main result now follows.

Theorem 5.9. *The isomorphism problem for automatic structures is Σ_1^1 -complete.* \square

REFERENCES

- [1] M. Benedikt and L. Libkin. Tree extension algebras: logics, automata, and query languages. In *Proceedings of the 17th Annual IEEE Symposium on Logic in Computer Science (LICS)*, pages 203–212, 2002.
- [2] M. Benedikt, L. Libkin, T. Schwentick and L. Segoufin. A Model-Theoretic approach to regular string relations. In *Proceedings of the 16th Annual IEEE Symposium on Logic in Computer Science (LICS)*, pages 431–440, 2001.
- [3] C. H. Bennett. Logical reversibility of computation. *IBM Journal of Research and Development*, 17(6):525–532, November 1973.
- [4] A. Blumensath. *Automatic Structures*. Diploma thesis, RWTH Aachen, 1999.
- [5] A. Blumensath and E. Grädel. Automatic structures. In *15th Symposium on Logic in Computer Science (LICS)*, pages 51–62, 2000.
- [6] J. W. Cannon, D. B. H. Epstein, D. F. Holt, S. V. F. Levy, M. S. Paterson and W. P. Thurston. *Word processing in groups*. Jones and Bartlett, 1992.
- [7] C. Delhommé. Automaticité des ordinaux et des graphes homogènes. *Comptes Rendus Mathématique*, 339(1):5–10, 2004.
- [8] S. Eilenberg, C. C. Elgot and J. C. Shepherdson. Sets recognised by n -tape automata. *Journal of Algebra*, 13(4):447–464, 1969.
- [9] S. S. Goncharov and J. F. Knight. Computable structure and non-structure theorems. *Algebra and Logic*, 41(6):351–373, 2002.
- [10] W. A. Hodges. *Model Theory*. Cambridge University Press, 1993.
- [11] H. Ishihara, B. Khoussainov and S. Rubin. Some results on automatic structures. In *17th Symposium on Logic in Computer Science (LICS)*, pages 235–242, 2002.
- [12] B. Khoussainov and A. Nerode. Automatic presentations of structures. *Lecture Notes in Computer Science*, 960:367–392, 1995.
- [13] B. Khoussainov, S. Rubin and F. Stephan. Automatic linear orders and trees. *ACM Transactions on Computational Logic (TOCL)*, 6(4), 2005.

- [14] L. Libkin and F. Neven. Logical definability and query languages over unranked trees. *Proceedings of the 18th Annual IEEE Symposium on Logic in Computer Science (LICS)*, pages 178–187, 2003.
- [15] A. Nies. Describing Groups. To appear, available at <http://www.cs.auckland.ac.nz/~nies/papers/>.
- [16] S. Rubin. *Automatic Structures*. Phd thesis, University of Auckland, 2004.

ORDER-INVARIANT MSO IS STRONGER THAN COUNTING MSO IN THE FINITE

TOBIAS GANZOW ¹ AND SASHA RUBIN ²

¹ Mathematische Grundlagen der Informatik, RWTH Aachen, Germany
E-mail address: ganzow@logic.rwth-aachen.de

² Department of Computer Science, University of Auckland, New Zealand
E-mail address: rubin@cs.auckland.ac.nz

ABSTRACT. We compare the expressiveness of two extensions of monadic second-order logic (MSO) over the class of finite structures. The first, counting monadic second-order logic (CMSO), extends MSO with first-order modulo-counting quantifiers, allowing the expression of queries like “the number of elements in the structure is even”. The second extension allows the use of an additional binary predicate, not contained in the signature of the queried structure, that must be interpreted as an arbitrary linear order on its universe, obtaining order-invariant MSO.

While it is straightforward that every CMSO formula can be translated into an equivalent order-invariant MSO formula, the converse had not yet been settled. Courcelle showed that for restricted classes of structures both order-invariant MSO and CMSO are equally expressive, but conjectured that, in general, order-invariant MSO is stronger than CMSO.

We affirm this conjecture by presenting a class of structures that is order-invariantly definable in MSO but not definable in CMSO.

1. Introduction

Linear orders play an important role in descriptive complexity theory since certain results relating the expressive power of logics to complexity classes, e.g., the Immerman-Vardi Theorem that LFP captures PTIME, only hold for classes of linearly ordered structures. Usually, the order only serves to systematically access all elements of the structure, and consequently to encode the configurations of a step-wise advancing computation of a Turing machine by tuples of elements of the structure. In these situations we do not actually want to make statements about the properties of the order, but merely want to have an arbitrary linear order available to express the respective coding techniques.

Furthermore, when actually working with finite structures in an algorithmic context, e.g., when evaluating queries in a relational database, we are in fact working on an implicitly ordered structure since, although relations in a database are modelled as *sets* of tuples, the relations are nevertheless stored as *ordered sequences* of tuples in memory or on a disk. As

1998 ACM Subject Classification: F.4.1 Mathematical Logic.

Key words and phrases: MSO, Counting MSO, order-invariance, expressiveness, Ehrenfeucht-Fraïssé game.



this linear order is always available (though, as in the case of databases, it is implementation-dependent and may even change over time as tuples are inserted or deleted), we could allow queries to make use of an additional binary predicate that is interpreted as a linear order on the universe of the structure, but require the outcome of the query not to depend on the actual ordering, but to be *order-invariant*. Precisely, given a τ -structure \mathfrak{A} , we allow queries built over an expanded vocabulary $\tau \dot{\cup} \{<\}$, and say that a query φ is *order-invariant* if $(\mathfrak{A}, <_1) \models \varphi \iff (\mathfrak{A}, <_2) \models \varphi$ for all possible relations $<_1$ and $<_2$ linearly ordering A .

Using Ehrenfeucht-Fraïssé-games for MSO, one can see that MSO on sets (i.e., structures over an empty vocabulary) is too weak to express that the universe contains an even number of elements. However, this is possible if the universe is linearly ordered: simply use the MSO sentence stating that the maximal element should be contained in the set of elements on even positions in the ordering. Obviously, such a sentence is order-invariant since rearranging the elements does not affect its truth value. Gurevich uses this observation to show that the property of Boolean algebras having an even number of atoms, although not definable in FO, is order-invariantly definable in FO (simulating the necessary MSO-quantification over sets of atoms by FO-quantification over the elements of the Boolean algebra).

If we explicitly add modulo-counting to MSO, e.g., via modulo-counting first-order quantifiers such as “there exists an even number of elements x such that ...”, we obtain *counting monadic second-order logic* (CMSO), and the question naturally arises as to whether there are properties not expressible in CMSO that can be expressed order-invariantly in MSO.

In fact, a second separation example due to Otto gives a hint in that direction. The class of structures presented in [Ott00] even separates order-invariant FO from FO extended by arbitrary unary generalised quantifiers, i.e., especially modulo-counting quantifiers, and exploits the idea of “hiding” a part of the structure such that it is only meaningfully usable for queries in presence of a linear order (or, as actually proven in the paper, in presence of an arbitrary choice function).

The expressiveness of CMSO has been studied, e.g., in [Cou90], where it is mainly compared to MSO, and in [Cou96] it is shown that, on the class of forests, order-invariant MSO is no more expressive than CMSO. As pointed out in [BS05], this can be generalised using results in [Lap98] to classes of structures of bounded tree-width. But still, this left open Courcelle’s conjecture: that order-invariant MSO is strictly stronger than CMSO for general graphs [Cou96, Conjecture 7.3].

In this paper, we present a suitable characterisation of CMSO-definability in terms of an Ehrenfeucht-Fraïssé game, and later, as the main contribution, we present a separating example showing that a special class of graphs is indeed definable by an order-invariant MSO sentence but not by a counting MSO sentence.

2. Preliminaries

Throughout the paper \mathbb{N} denotes the set of non-negative integers and $\mathbb{N}^+ := \mathbb{N} - \{0\}$. Given a non-empty finite set $M = \{m_1, \dots, m_k\} \subseteq_{\text{fin}} \mathbb{N}^+$, let $\text{lcm}(M) := \text{lcm}(m_1, \dots, m_k)$ denote the least common multiple of all elements in M ; additionally, we define $\text{lcm}(\emptyset) = 1$. For sets X and Y as well as M as before, we abbreviate that $|X| \equiv |Y| \pmod{m}$ for all $m \in M$ by using the shorthand $|X| \equiv |Y| \pmod{M}$.

We restrict our attention to finite τ -structures with a nonempty universe over a countable relational vocabulary τ , possibly with constants, and we will mainly deal with monadic second-order logic and some of its extensions. For more details concerning finite model theory, we refer to [EF95] or [Lib04].

When comparing the expressiveness of two logics \mathcal{L} and \mathcal{L}' , we say that \mathcal{L}' is at least as expressive as \mathcal{L} , denoted $\mathcal{L} \subseteq \mathcal{L}'$, if for every $\varphi \in \mathcal{L}[\tau]$ there exists a $\varphi' \in \mathcal{L}'[\tau]$ such that $\text{Mod}(\varphi) = \text{Mod}(\varphi')$, where $\text{Mod}(\varphi)$ denotes the class of all finite τ -structures satisfying φ .

2.1. Counting MSO

The notion of (modulo-)counting monadic second-order logic (CMSO) can be introduced in two different, but nonetheless equivalent, ways. The first view of CMSO is via an extension of MSO by modulo-counting first-order quantifiers.

Definition 2.1. Let τ be a signature and $M \subseteq \mathbb{N}^+$ a set of moduli, then

- every formula $\varphi \in \text{MSO}[\tau]$ is also a formula in $\text{CMSO}^{(M)}[\tau]$, and
- if $\varphi(x) \in \text{CMSO}^{(M)}[\tau]$ and $m \in M$, then $\exists^{(m)}x.\varphi(x) \in \text{CMSO}^{(M)}[\tau]$.

If we do not restrict the set of modulo-counting quantifiers being used, we get the full language $\text{CMSO}[\tau] = \text{CMSO}^{(\mathbb{N}^+)}[\tau]$. The semantics of MSO formulae is as expected, and we have $\mathfrak{A} \models \exists^{(m)}x.\varphi(x)$ if and only if $|\{a \in A : \mathfrak{A} \models \varphi(a)\}| \equiv 0 \pmod{m}$. The quantifier rank $\text{qr}(\psi)$ of a $\text{CMSO}[\tau]$ formula ψ is defined as for MSO-formulae with the additional rule that $\text{qr}(\exists^{(m)}x.\varphi(x)) = 1 + \text{qr}(\varphi)$, i.e., we do not distinguish between different kinds of quantifiers.

In this paper we use an alternative but equivalent definition of CMSO, namely the extension of the MSO language by monadic second-order predicates $C^{(m)}$ which hold true of a set X if and only if $|X| \equiv 0 \pmod{m}$. As in the definition above, formulae of the fragment $\text{CMSO}^{(M)}[\tau]$ may only use predicates $C^{(m)}$ where $m \in M$. The back-and-forth translation can be carried out along the following equivalences which increase the quantifier rank by at most one in each step:

$$\begin{aligned}\exists^{(m)}x.\varphi(x) &\equiv \exists X(C^{(m)}(X) \wedge \forall x(Xx \leftrightarrow \varphi(x))) \quad \text{and} \\ C^{(m)}(X) &\equiv \exists^{(m)}x.Xx.\end{aligned}$$

Furthermore, the introduction of additional predicates $C^{(m,r)}$ (or, equivalently, additional modulo-counting quantifiers $\exists^{(m,r)}$) stating for a set X that $|X| \equiv r \pmod{m}$ does not increase the expressive power since they can be simulated as follows (with only a constant increase of quantifier rank):

$$C^{(m,r)}(X) \equiv \exists X_0("X_0 \subseteq X" \wedge "|X_0| = r" \wedge "C^{(m)}(X \setminus X_0)"),$$

where all subformulae are easily expressible in MSO.

Later, we will introduce an Ehrenfeucht-Fraïssé game capturing the expressiveness of CMSO with this extended set of second-order predicates.

2.2. Order-invariance

Let τ be a relational vocabulary and $\varphi \in \text{MSO}[\tau \dot{\cup} \{<\}]$, i.e., φ may contain an additional relation symbol $<$. Then φ is called *order-invariant on a class \mathcal{C} of τ -structures* if, and only if, $(\mathfrak{A}, <_1) \models \varphi \iff (\mathfrak{A}, <_2) \models \varphi$ for all $\mathfrak{A} \in \mathcal{C}$ and all linear orders $<_1$ and $<_2$ on A .

Although, in general, it is undecidable whether a given MSO-formula is order-invariant in the finite, we will speak of the *order-invariant fragment of MSO*, denoted by $\text{MSO}[<]_{inv}$, that contains all formulae that are order-invariant on the class of all finite structures.

It is an easy observation that every CMSO formula is equivalent over the class of all finite structures to an order-invariant MSO formula by translating counting quantifiers in the following way:

$$\exists^{(q)} x. \varphi(x) := \exists X \exists X_0 \dots \exists X_{q-1} \left(\begin{array}{l} \forall x (Xx \leftrightarrow \varphi(x)) \wedge \text{"}\{X_0, \dots, X_{q-1}\}\text{ is a partition of } X\text{"} \\ \wedge \exists x (X_0 x \wedge \forall y (Xy \rightarrow x \leq y)) \wedge \exists x (X_{q-1} x \wedge \forall y (Xy \rightarrow x \geq y)) \\ \wedge \forall x \forall y \left(S_{\varphi, <} (x, y) \rightarrow \left(\bigwedge_{i=0}^{q-1} X_i x \leftrightarrow X_{i+1} \pmod{q} y \right) \right) \end{array} \right)$$

where $S_{\varphi, <}$ defines the successor relation induced by an arbitrary order $<$ on the universe of the structure restricted to the set X of elements for which φ holds.

Note that the quantifier rank of the translated formula is not constant but bounded by the parameter in the counting quantifier.

3. An Ehrenfeucht-Fraïssé game for CMSO

The Ehrenfeucht-Fraïssé game capturing expressiveness of MSO parameterised by the quantifier-rank (cf. [EF95, Lib04]) can be naturally extended to a game capturing the expressiveness of CMSO parameterised by the quantifier rank and the set of moduli being used in the cardinality predicates or counting quantifiers.

Viewing CMSO as MSO with additional quantifiers $\exists^{(m)} x. \varphi(x)$ for all m in a fixed set M leads to a new type of move described, e.g., in the context of extending FO by modulo-counting quantifiers in [Nur00]. Since a modulo-counting quantifier actually combines notions of a first-order and a monadic second-order quantifier in the sense that it makes a statement about the cardinality of a certain *set* of elements, but on the other hand, it behaves like a first-order quantifier binding an *element* variable and making a statement about that particular element, the move capturing modulo-counting quantification consists of two phases. First, Spoiler and Duplicator select sets of elements S and D in the structures such that $|S| \equiv |D| \pmod{M}$, and in the second phase, Spoiler and Duplicator select elements a and b such that $a \in S$ if and only if $b \in D$. After the move, reflecting the first-order nature of the quantifier, only the two selected elements a and b are remembered and contribute to the next position in the game, whereas the information about the chosen sets is discarded.

We prefer viewing CMSO via second-order cardinality predicates, yielding an Ehrenfeucht-Fraïssé game that allows a much clearer description of winning strategies. Since we do not have additional quantifiers, we have exactly the same types of moves as in the Ehrenfeucht-Fraïssé game for MSO, and we merely modify the winning condition to take the new predicates into account.

Towards this end, we first introduce a suitable concept of partial isomorphisms between structures.

Definition 3.1. With any structure \mathfrak{A} and any set $M \subseteq_{\text{fin}} \mathbb{N}^+$ we associate the (first-order) power set structure $\mathfrak{A}^M := (\mathcal{P}(A), (C^{(m,r)})_{\substack{m \in M \\ 0 \leq r < m}})$, where the predicates $C^{(m,r)}$ are interpreted in the obvious way. (Note that first-order predicates in the power set structure \mathfrak{A}^M naturally correspond to second-order predicates in \mathfrak{A} .)

Let \mathfrak{A} and \mathfrak{B} be τ -structures, and let $M \subseteq_{\text{fin}} \mathbb{N}^+$ be a fixed set of moduli. Then the mapping $(A_1, \dots, A_s, a_1, \dots, a_t) \mapsto (B_1, \dots, B_s, b_1, \dots, b_t)$ is called a *twofold partial isomorphism between \mathfrak{A} and \mathfrak{B} with respect to M* if

- (i) $(a_1, \dots, a_t) \mapsto (b_1, \dots, b_t)$ is a partial isomorphism between $(\mathfrak{A}, A_1, \dots, A_s)$ and $(\mathfrak{B}, B_1, \dots, B_s)$ and
- (ii) $(A_1, \dots, A_s) \mapsto (B_1, \dots, B_s)$ is a partial isomorphism between \mathfrak{A}^M and \mathfrak{B}^M .

We propose the following Ehrenfeucht-Fraïssé game to capture the expressiveness of CMSO where the use of moduli is restricted to a (finite) set M and formulae of quantifier rank at most r .

Definition 3.2 (Ehrenfeucht-Fraïssé game for CMSO). Let $M \subseteq_{\text{fin}} \mathbb{N}^+$ and $r \in \mathbb{N}$. The r -round (mod M) Ehrenfeucht-Fraïssé game $\mathcal{G}_r^M(\mathfrak{A}, \mathfrak{B})$ is played by Spoiler and Duplicator on τ -structures \mathfrak{A} and \mathfrak{B} . In each turn, Spoiler can choose between the following types of moves:

- *point move*: Spoiler selects an element in one of the structures, and Duplicator answers by selecting an element in the other structure.
- *set move*: Spoiler selects a set of elements X in one of the structures, and Duplicator responds by choosing a set of elements Y in the other structure.

After $r = s + t$ rounds, when the players have chosen sets A_1, \dots, A_s and B_1, \dots, B_s as well as elements a_1, \dots, a_t and b_1, \dots, b_t in an arbitrary order, Duplicator wins the game if, and only if, $(A_1, \dots, A_s, a_1, \dots, a_t) \mapsto (B_1, \dots, B_s, b_1, \dots, b_t)$ is a twofold partial isomorphism between \mathfrak{A} and \mathfrak{B} with respect to M .

First note that, although Duplicator is required to answer a set move X by a set Y such that $|X| \equiv |Y| \pmod{M}$ in order to win, we do not have to make this explicit in the rules of the moves since these cardinality constraints are already imposed by the winning condition (X and Y would not define a twofold partial isomorphism if they did not satisfy the same cardinality predicates). Furthermore, for $M = \emptyset$ or $M = \{1\}$, the resulting game $\mathcal{G}_r^M(\mathfrak{A}, \mathfrak{B})$ corresponds exactly to the usual Ehrenfeucht-Fraïssé game for MSO.

Theorem 3.3. Let \mathfrak{A} and \mathfrak{B} be τ -structures, $r \in \mathbb{N}$, and $M \subseteq_{\text{fin}} \mathbb{N}$. Then the following are equivalent:

- (i) $\mathfrak{A} \equiv_r^M \mathfrak{B}$, i.e., $\mathfrak{A} \models \varphi$ if and only if $\mathfrak{B} \models \varphi$ for all $\varphi \in \text{CMSO}^{(M)}[\tau]$ with $\text{qr}(\varphi) \leq r$.
- (ii) Duplicator has a winning strategy in the r -round (mod M) Ehrenfeucht-Fraïssé game $\mathcal{G}_r^M(\mathfrak{A}, \mathfrak{B})$. ■

To prove non-definability results, we can make use of the following standard argument.

Proposition 3.4. A class \mathcal{C} of τ -structures is not definable in CMSO if, for every $r \in \mathbb{N}$ and every $M \subseteq_{\text{fin}} \mathbb{N}^+$, there are τ -structures $\mathfrak{A}_{M,r}$ and $\mathfrak{B}_{M,r}$ such that $\mathfrak{A}_{M,r} \in \mathcal{C}$, $\mathfrak{B}_{M,r} \notin \mathcal{C}$, and $\mathfrak{A}_{M,r} \equiv_r^M \mathfrak{B}_{M,r}$.

The following lemma, stating that the CMSO-theory of disjoint unions can be deduced from the CMSO-theories of the components, can either be proved, as carried out in [Cou90, Lemma 4.5], by giving an effective translation of sentences talking about the disjoint union of two structures into a Boolean combination of sentences each talking about the individual structures, or by using a game-oriented view showing that winning strategies for Duplicator in the games on two pairs of structures can be combined into a winning strategy on the pair of disjoint unions of the structures.

Lemma 3.5. *Let $\mathfrak{A}_1, \mathfrak{A}_2, \mathfrak{B}_1$, and \mathfrak{B}_2 be τ -structures such that $\mathfrak{A}_1 \equiv_r^M \mathfrak{B}_1$ and $\mathfrak{A}_2 \equiv_r^M \mathfrak{B}_2$. Then $\mathfrak{A}_1 \dot{\cup} \mathfrak{A}_2 \equiv_r^M \mathfrak{B}_1 \dot{\cup} \mathfrak{B}_2$.*

Proof. Consider the game on $\mathfrak{A} := \mathfrak{A}_1 \dot{\cup} \mathfrak{A}_2$ and $\mathfrak{B} := \mathfrak{B}_1 \dot{\cup} \mathfrak{B}_2$. A Spoiler's point move in \mathfrak{A} (resp., in \mathfrak{B}) is answered by Duplicator according to her winning strategy in either $\mathcal{G}_r^M(\mathfrak{A}_1, \mathfrak{B}_1)$ or $\mathcal{G}_r^M(\mathfrak{A}_2, \mathfrak{B}_2)$. A set move $S \subseteq A$ (analogous for $S \subseteq B$) is decomposed into two subsets $S_1 := S \cap A_1$ and $S_2 := S \cap A_2$, and is answered by Duplicator by the set $D := D_1 \cup D_2$ consisting of the sets D_1 and D_2 chosen according to her winning strategies as responses to S_1 and S_2 in the respective games $\mathcal{G}_r^M(\mathfrak{A}_1, \mathfrak{B}_1)$ and $\mathcal{G}_r^M(\mathfrak{A}_2, \mathfrak{B}_2)$.

Since A_1 and A_2 as well as B_1 and B_2 are disjoint, we have $|S| = |S_1| + |S_2|$ and $|D| = |D_1| + |D_2|$. Furthermore, $|S_1| \equiv |D_1| \pmod{M}$ and $|S_2| \equiv |D_2| \pmod{M}$ as the sets D_1 and D_2 are chosen according to Duplicator's winning strategies in the games on \mathfrak{A}_1 and \mathfrak{B}_1 , and \mathfrak{A}_2 and \mathfrak{B}_2 , respectively. Since $\equiv \pmod{M}$ is a congruence relation with respect to addition, we have that $|S| \equiv |D| \pmod{M}$. It is easily verified that the sets and elements chosen according to this strategy indeed define a twofold partial isomorphism between \mathfrak{A} and \mathfrak{B} . ■

As a direct corollary we obtain the following result that will be used in the inductive step in the forthcoming proofs.

Corollary 3.6. *Let $\mathfrak{A}_1, \mathfrak{A}_2, \mathfrak{B}_1$, and \mathfrak{B}_2 be τ -structures, such that $\mathfrak{A}_1 \equiv_r^M \mathfrak{B}_1$ and $\mathfrak{A}_2 \equiv_r^M \mathfrak{B}_2$. Then $(\mathfrak{A}_1 \dot{\cup} \mathfrak{A}_2, A_1) \equiv_r^M (\mathfrak{B}_1 \dot{\cup} \mathfrak{B}_2, B_1)$.*

Proof. We consider the following $\tau \dot{\cup} \{P\}$ -expansions of the given structures: $\mathfrak{A}'_1 := (\mathfrak{A}_1, A_1)$, $\mathfrak{B}'_1 := (\mathfrak{B}_1, B_1)$, $\mathfrak{A}'_2 := (\mathfrak{A}_2, \emptyset)$, and $\mathfrak{B}'_2 := (\mathfrak{B}_2, \emptyset)$. It is immediate that

- (i) $\mathfrak{A}_1 \equiv_r^M \mathfrak{B}_1$ implies $(\mathfrak{A}_1, A_1) \equiv_r^M (\mathfrak{B}_1, B_1)$, and
- (ii) $\mathfrak{A}_2 \equiv_r^M \mathfrak{B}_2$ implies $(\mathfrak{A}_2, \emptyset) \equiv_r^M (\mathfrak{B}_2, \emptyset)$

since Duplicator can obviously win the respective Ehrenfeucht-Fraïssé games on the expanded structures using the same strategies as in the games proving the equivalences on the left-hand side. The claim follows by applying the previous lemma to the $\tau \dot{\cup} \{P\}$ -expansions. ■

It is well known that MSO exhibits a certain weakness regarding the ability to specify cardinality constraints on sets, i.e., structures over an empty vocabulary. A proof of this fact using Ehrenfeucht-Fraïssé games can be found in [Lib04]. By adapting this proof, we show that this is still the case for CMSO.

Lemma 3.7. *Let \mathfrak{A} and \mathfrak{B} be \emptyset -structures, $M \subseteq_{fin} \mathbb{N}^+$, and $r \in \mathbb{N}$. Then $\mathfrak{A} \equiv_r^M \mathfrak{B}$ if $|A|, |B| \geq (2^{r+1} - 4) \text{lcm}(M)$ and $|A| \equiv |B| \pmod{M}$.*

Proof. We prove by induction on the number of rounds that Duplicator wins the $(\text{mod } M)$ r -round Ehrenfeucht-Fraïssé game $\mathcal{G}_r^M(\mathfrak{A}, \mathfrak{B})$. For $r = 0$ and $r = 1$ the claim is obviously

true. Let $r > 1$, assume that the claim holds for $r - 1$, and consider the first move of the r -round game. We assume that Spoiler makes his move in \mathfrak{A} since the reasoning in the other case is completely symmetric.

If Spoiler makes a set move $S \subseteq A$, we consider the following cases:

- (1) $|S| < (2^r - 4) \cdot \text{lcm}(M)$ (or $|A - S| < (2^r - 4) \cdot \text{lcm}(M)$). Then Duplicator selects a set $D \subseteq B$ such that $|D| = |S|$ (or $|B - D| = |A - S|$), and hence $S \cong D$ and $A - S \equiv_{r-1}^M B - D$ (or $A - S \cong B - D$ and $S \equiv_{r-1}^M D$).
- (2) $|S|, |A - S| \geq (2^r - 4) \cdot \text{lcm}(M)$. Then Duplicator selects a set $D \subseteq B$ such that $|D| \equiv |S| \pmod{M}$ and $|D|, |B - D| \geq (2^r - 2) \cdot \text{lcm}(M)$. In fact, she chooses for D half of the elements and chooses $\ell < \text{lcm}(M)$ additional ones to fulfil the cardinality constraints $|D| \equiv |S| \pmod{M}$. Then, for the set $B - D$ of non-selected elements, we have

$$\begin{aligned} |B - D| &\geq \frac{1}{2}((2^{r+1} - 4) \text{lcm}(M)) - \ell \geq (2^r - 2) \text{lcm}(M) - \text{lcm}(M) \\ &\geq (2^r - 4) \text{lcm}(M) \end{aligned}$$

for all ℓ satisfying $0 \leq \ell < \text{lcm}(M)$. Since $|D| = |B - D| + 2\ell$, obviously $|D| \geq (2^r - 4) \text{lcm}(M)$ as well.

Thus, in both cases, by the induction hypothesis we get $S \equiv_{r-1}^M D$ and $A - S \equiv_{r-1}^M B - D$. Hence, by Corollary 3.6 $(A, S) \equiv_{r-1}^M (B, D)$, i.e., Duplicator has a winning strategy in the remaining $(r - 1)$ -round game from position (S, D) .

If Spoiler makes a point move $s \in A$, Duplicator answers by choosing an arbitrary element $d \in B$. Similar to Case 1 above, we observe that $(\{s\}, s) \cong (\{d\}, d)$ and $A - \{s\} \equiv_{r-1}^M B - \{d\}$ by the induction hypothesis. Thus, by Lemma 3.5, $(A, s) \equiv_{r-1}^M (B, d)$ implying that Duplicator has a winning strategy for the remaining $r - 1$ rounds from position (s, d) . ■

4. The Separating Example

We will first give a brief description of our example showing that $\text{MSO}[\langle\rangle_{\text{inv}}$ is strictly more expressive than CMSO . We consider a property of two-dimensional grids, namely that the vertical dimension divides the horizontal dimension. This property is easily definable in MSO for grids that are given as directed graphs with two edge relations, one for the horizontal edges pointing rightwards, and one for the vertical edges pointing upwards, by defining a new relation of diagonal edges combining one step rightwards and one step upwards wrapping around from the top border to the bottom border but not from the right to the left border. Note that there is a path following those diagonal edges starting from the bottom-left corner of the grid and ending in the top-right corner if, and only if, the vertical dimension divides the horizontal dimension of the grid. Thus, for our purposes, we have to weaken the structure in the sense that we hide information that remains accessible to $\text{MSO}[\langle\rangle_{\text{inv}}$ -formulae but not to CMSO formulae.

An appropriate loss of information is achieved by replacing the two edge relations with their reflexive symmetric transitive closure, i.e., we consider grids as structures with two equivalence relations which provide a notion of *rows* and *columns* of the grid. Obviously, notions like corner and border vertices as well as the notion of an order on the rows and columns that were important for the MSO -definition of the divisibility property are lost, but clearly, all these notions can be regained in presence of an order. First, the order allows us to uniquely define an element (e.g. the $<$ -least element) to be the bottom-left corner of

the grid, and second, the order induces successor relations on the set of columns and the set of rows, from which both horizontal and vertical successor vertices of any vertex can be deduced. Since the divisibility property is obviously invariant with respect to the ordering of the rows or columns, this allows for expressing it in $\text{MSO}[\langle\rangle_{\text{inv}}]$. In the course of this section we will develop the arguments showing that CMSO fails to express this property on the following class of grid-like structures.

Definition 4.1. A *cliquey* (k, ℓ) -grid is a $\{\sim_h, \sim_v\}$ -structure that is isomorphic to $\mathfrak{G}_{k\ell} := (\{0, \dots, k-1\} \times \{0, \dots, \ell-1\}, \sim_h, \sim_v)$, where

$$\begin{aligned}\sim_h &:= \{((x, y), (x', y')) : x = x'\} \text{ and} \\ \sim_v &:= \{((x, y), (x', y')) : y = y'\},\end{aligned}$$

i.e., \sim_h consists of exactly k equivalence classes (called *rows*), each containing ℓ elements, and \sim_v consists of exactly ℓ equivalence classes (called *columns*), each containing k elements, such that every equivalence class of \sim_h intersects every equivalence class of \sim_v in exactly one element and vice versa.

A *horizontally coloured cliquey* (k, ℓ) -grid, denoted $\mathfrak{G}_{k\ell}^{\text{col}}$, is the expansion of the $\{\sim_v\}$ -reduct of the cliquey grid $\mathfrak{G}_{k\ell}$ by unary predicates $\{P_1, \dots, P_k\}$, where the information of \sim_h is retained in the k new predicates (in the following referred to as *colours*) such that each set P_i corresponds to exactly one former equivalence class.

Note that the same class of grid-like structures has already been used by Otto in a proof showing that the number of monadic second-order quantifiers gives rise to a strict hierarchy over finite structures [Ott95].

The class is first-order definable by a sentence ψ_{grid} stating that

- \sim_v and \sim_h are equivalence relations, and
- every pair consisting of one equivalence class of \sim_h and \sim_v each has exactly one element in common

as these properties are sufficient to enforce the desired grid-like structure. Note that even the second property is first-order definable since every equivalence class is uniquely determined by each of its elements.

The following two lemmata justify the introduction of the notion of horizontally coloured cliquey grids for use in the forthcoming proofs.

Lemma 4.2. Let $\mathfrak{G}_{k\ell_1}^{\text{col}}$, $\mathfrak{G}_{k\ell_2}^{\text{col}}$, $\mathfrak{G}_{k\ell'_1}^{\text{col}}$, and $\mathfrak{G}_{k\ell'_2}^{\text{col}}$ be horizontally coloured cliquey grids such that $\mathfrak{G}_{k\ell_1}^{\text{col}} \equiv_r^M \mathfrak{G}_{k\ell'_1}^{\text{col}}$ and $\mathfrak{G}_{k\ell_2}^{\text{col}} \equiv_r^M \mathfrak{G}_{k\ell'_2}^{\text{col}}$. Then $\mathfrak{G}_{k, \ell_1 + \ell_2}^{\text{col}} \equiv_r^M \mathfrak{G}_{k, \ell'_1 + \ell'_2}^{\text{col}}$.

Proof. Note that, since there are no horizontal edges in horizontally coloured cliquey grids and the vertical dimension of all grids is k , $\mathfrak{G}_{k, \ell_1 + \ell_2}^{\text{col}}$ is the disjoint union of the two smaller horizontally coloured cliquey grids $\mathfrak{G}_{k\ell_1}^{\text{col}}$ and $\mathfrak{G}_{k\ell_2}^{\text{col}}$, and of course, the same holds for $\mathfrak{G}_{k, \ell'_1 + \ell'_2}^{\text{col}}$. Thus, the claim follows by Lemma 3.5. ■

Lemma 4.3. Let $\mathfrak{G}_{k\ell}^{\text{col}} \equiv_r^M \mathfrak{G}_{k\ell'}^{\text{col}}$. Then $\mathfrak{G}_{k\ell} \equiv_r^M \mathfrak{G}_{k\ell'}$.

Proof. For each fixed horizontal dimension k , there exists a one-dimensional quantifier-free interpretation of a cliquey grid in its respective horizontally coloured counterpart since we can define the horizontal equivalence relation \sim_h in terms of the colours as follows:

$$x \sim_h y \equiv \bigvee_{i=1}^k P_i x \wedge P_i y.$$

■

Actually, the argument implies that Duplicator wins a game on cliquey grids using the same strategy that is winning in the corresponding game on coloured grids since a strategy preserving the colours of selected elements especially preserves the equivalence relation \sim_h .

Before stating the main lemma, we will first prove a combinatorial result which will later help Duplicator in synthesising her winning strategy and introduce the following weakened notion of equality between numbers.

Definition 4.4. Two numbers $a, b \in \mathbb{N}$ are called *threshold t equal (mod M)*, denoted $a =_t^M b$, if

- (i) $a = b$ or
- (ii) $a, b \geq t$ and $a \equiv b \pmod{M}$.

Intuitively, $a =_t^M b$ means that the numbers are equal if they are small, or that they are at least congruent modulo all $m \in M$ if they are both at least as large as the threshold t .

Lemma 4.5. For every $p, t \in \mathbb{N}$, and $M \subseteq_{fin} \mathbb{N}^+$, we can choose an arbitrary $T \geq p \cdot (t + \text{lcm}(M) - 1)$ such that for all sets A and B with $|A| =_T^M |B|$ and for every equivalence relation \approx_A on A of index at most p there exists an equivalence relation \approx_B on B and a bijection $g: A/\approx_A \rightarrow B/\approx_B$ satisfying $|\{a' \in A : a \approx_A a'\}| =_t^M |g(\{a' \in A : a \approx_A a'\})|$ for all $a \in A$.

Proof. We let $\{a_1, \dots, a_{p'}\}$, where $p' \leq p$ denotes the index of \approx_A , be the set of class representatives of A/\approx_A , and we let $[a]_{\approx_A} := \{a' \in A : a' \approx_A a\}$ denote the equivalence class of a in A . Note that we will usually omit the subscript \approx_A if it is clear from the context and instead reserve the letters a and b for elements denoting equivalence classes in A and B , respectively. Furthermore, a set will be called *small* in the following if it contains less than t elements and *large* otherwise.

The equivalence relation \approx_B on B is constructed by partitioning the set into p' disjoint non-empty subsets $\{B_1, \dots, B_{p'}\}$ as follows. If $|A| = |B|$, for each class $[a_i]$, we choose a set B_i with exactly $|[a_i]|$ many elements. If $|A|, |B| \geq T$, we have to distinguish between the treatment of small and large classes. Since $|A| \geq T \geq p \cdot (t + \text{lcm}(M) - 1)$, $\text{lcm}(M) \geq 1$, and the index of \approx_A is at most p , at least one of the equivalence classes contains at least t elements, i.e., it is large, and without loss of generality, it is assumed that this is the case for $[a_1]$. For each small class $[a_i]$, we choose a set B_i with exactly $|[a_i]|$ many elements. If $[a_i]$ is large, we choose a set B_i containing $t + \ell$ many elements where ℓ is the smallest non-negative integer such that $|[a_i]| \equiv |B_i| \pmod{M}$. The number of elements selected according to these rules is at most $p \cdot (t + \text{lcm}(M) - 1) \leq T \leq |B|$. Since $[a_1]$ is large by assumption, any possibly remaining elements in B , that have not been assigned to one of the subsets $B_1, \dots, B_{p'}$ yet, can be safely added to B_1 without violating the condition that $|[a_1]| \equiv |B_1| \pmod{M}$.

This partitioning uniquely defines the equivalence relation $\approx_B := \bigcup_{i=1}^{p'} (B_i \times B_i)$ on B . By selecting an arbitrary element of each B_i we get a set of class representatives $\{b_1, \dots, b_{p'}\}$ which directly yields the bijection $g: [a_i] \mapsto [b_i]$ for all $1 \leq i \leq p'$ satisfying $|[a]| =_t^M |g([a])|$ for all $a \in A$ by construction. ■

The following lemma extends the results on CMSO-equivalence of *large enough sets* to *large enough grids* by giving a sufficient condition on the sizes of two grids for the existence of a winning strategy for Duplicator in an r -round (mod M) game on the two structures. Due to the inductive nature of the proof that involves, in each step, a construction of

equivalence classes as in the above lemma, we need as a criterion for the size, for fixed $p \in \mathbb{N}$ and $M \subseteq_{\text{fin}} \mathbb{N}^+$, a function $f_{p,M} : \mathbb{N} \rightarrow \mathbb{N}$ such that, for all $r \in \mathbb{N}^+$ and $t = f_{p,M}(r-1)$, we can choose $T = f_{p,M}(r)$ in the previous lemma. One function satisfying, for all $r \in \mathbb{N}^+$, the inequality $f_{p,M}(r) \geq p \cdot (f_{p,M}(r-1) + \text{lcm}(M) - 1)$ derived from the condition imposed on T is $f_{p,M}(r) = 2 \cdot (p^r - 1) \cdot \text{lcm}(M)$.

Lemma 4.6. *Let $M \subseteq_{\text{fin}} \mathbb{N}^+$, $r \in \mathbb{N}$ and $k > 1$ be fixed. Then for $f(r) := f_{2^k,M}(r) = (2^{kr+1} - 2) \text{lcm}(M)$, as given above, $\mathfrak{G}_{k\ell_1} \equiv_r^M \mathfrak{G}_{k\ell_2}$ if $\ell_1 =_{f(r)}^M \ell_2$.*

Proof. As motivated by Lemma 4.3, we consider the r -round $(\text{mod } M)$ Ehrenfeucht-Fraïssé game on the corresponding horizontally coloured cliquey grids $\mathfrak{G}_{k\ell_1}^{\text{col}}$ and $\mathfrak{G}_{k\ell_2}^{\text{col}}$, and we show by induction on the number of rounds that Duplicator has a winning strategy in this game.

Intuitively, the proof proceeds as follows. Spoiler's set move induces an equivalence relation on the set of columns forming the grid he plays in, and the previous lemma implies that Duplicator is able to construct an equivalence relation on the columns of the other grid which is similar in the sense that corresponding equivalence classes satisfy certain cardinality constraints. Since the grids can be regarded as disjoint unions of these equivalence classes, we can argue by induction that corresponding subparts of the two grids, being similar enough, cannot be distinguished during the remaining $r-1$ rounds of the game.

The case where $\ell_1 = \ell_2$ is trivial since grids of the same dimensions are isomorphic. Thus, we assume in the following that $\ell_1, \ell_2 \geq f(r)$ and $\ell_1 \equiv \ell_2 \pmod{M}$. The claim is obviously true for $r=0$, hence we assume that it holds for $r-1$ and proceed with the inductive step. As before, we assume without loss of generality that Spoiler makes his moves in $\mathfrak{G}_{k\ell_1}$ since the other case is symmetric.

A *coloured k -column* is a $\{\sim_v, P_1, \dots, P_k\}$ -structure isomorphic to $\mathfrak{C}_k^{\text{col}} := \mathfrak{G}_{k,1}^{\text{col}}$, such that a coloured grid can be regarded as a disjoint union of columns. Given a subset S of vertices of a grid and one of its coloured k -columns \mathfrak{C} with universe C , the *colour-type of \mathfrak{C} induced by S* is defined as the isomorphism type of the expansion $(\mathfrak{C}, S \cap C)$ denoted by $\text{tp}(\mathfrak{C}, S)$. Given a set \mathcal{F} of k -columns, each subset S of all of their vertices gives rise to an equivalence relation \approx_S on \mathcal{F} by virtue of $\mathfrak{C}_1 \approx_S \mathfrak{C}_2$ if, and only if, $\text{tp}(\mathfrak{C}_1, S) = \text{tp}(\mathfrak{C}_2, S)$. Note that the index of \approx_S is at most 2^k .

Assume, Spoiler performs a set move and chooses a subset S in $\mathfrak{G}_{k\ell_1}^{\text{col}} = \mathfrak{C}_1 \dot{\cup} \dots \dot{\cup} \mathfrak{C}_{\ell_1}$. As described above, S induces an equivalence relation \approx_S with at most 2^k equivalence classes on the set $\mathcal{F} = \{\mathfrak{C}_1, \dots, \mathfrak{C}_{\ell_1}\}$ of columns forming the grid. For $p = 2^k$, $t = f(r-1)$ and M as given, by the previous lemma, there is an equivalence relation \approx'_S on the set $\mathcal{F}' = \{\mathfrak{C}'_1, \dots, \mathfrak{C}'_{\ell_2}\}$ of columns on the Duplicator's grid $\mathfrak{G}_{k\ell_2}^{\text{col}}$ since $\ell_1, \ell_2 \geq f(r)$. Furthermore, there is a bijection g mapping equivalence classes of columns in one grid to the other.

Given that the index of both \approx_S and \approx'_S is $p' \leq p = 2^k$, we can assume $\{\mathfrak{C}_1, \dots, \mathfrak{C}_{p'}\}$ and $\{\mathfrak{C}'_1, \dots, \mathfrak{C}'_{p'}\}$ to be the sets of class representatives of \approx_S and \approx'_S , respectively. Duplicator now selects the unique set D of elements such that $\text{tp}(\mathfrak{C}, S) = \text{tp}(\mathfrak{C}', D)$ for all $1 \leq i \leq p'$, $\mathfrak{C} \in [\mathfrak{C}_i]$ and $\mathfrak{C}' \in g([\mathfrak{C}_i])$.

For each $1 \leq i \leq p'$, we let $\langle \mathfrak{C}_i \rangle := \mathfrak{G}_{k\ell_1}^{\text{col}} \upharpoonright [\mathfrak{C}_i]$ and $\langle \mathfrak{C}'_i \rangle := \mathfrak{G}_{k\ell_2}^{\text{col}} \upharpoonright [\mathfrak{C}'_i]$ denote the substructures of the grids $\mathfrak{G}_{k\ell_1}^{\text{col}}$ and $\mathfrak{G}_{k\ell_2}^{\text{col}}$ induced by the sets of columns $[\mathfrak{C}_i]$ and $[\mathfrak{C}'_i]$, respectively. By construction, we have $||[\mathfrak{C}_i]|| =_{f(r-1)}^M ||[\mathfrak{C}'_i]||$ for all i . Thus, depending on whether $[\mathfrak{C}_i]$ (and hence $[\mathfrak{C}'_i]$) are small or large with respect to the threshold $f(r-1)$, either $\langle \mathfrak{C}_i \rangle \cong \langle \mathfrak{C}'_i \rangle$ or $\langle \mathfrak{C}_i \rangle \equiv_{r-1}^M \langle \mathfrak{C}'_i \rangle$ by the induction hypothesis. Since S and D induce the

same colour-types on the columns in $[\mathfrak{C}_i]$ and $[\mathfrak{C}'_i]$, respectively, we have

$$(\langle \mathfrak{C}_i \rangle, S \cap \text{univ}(\langle \mathfrak{C}_i \rangle)) \equiv_{r-1}^M (\langle \mathfrak{C}'_i \rangle, D \cap \text{univ}(\langle \mathfrak{C}'_i \rangle))$$

for all i , where $\text{univ}(\cdot)$ denotes the universe of the respective structure. Thus, iterating Lemma 3.5 yields that Duplicator has a winning strategy in the remaining rounds of the game $\mathcal{G}_{r-1}^M(\mathfrak{G}_{k\ell_1}^{\text{col}}, \mathfrak{G}_{k\ell_2}^{\text{col}})$ from position (S, D) .

If Spoiler makes a point move s , say in column \mathfrak{C}_1 of the grid $\mathfrak{G}_{k\ell_1}^{\text{col}}$, Duplicator picks an arbitrary element d of the same colour in her grid, say in column \mathfrak{C}'_1 . As the substructures consisting of just the columns containing the chosen elements are isomorphic, i.e., $(\mathfrak{C}_1, s) \cong (\mathfrak{C}'_1, d)$, and by the induction hypothesis we have $\mathfrak{C}_2 \dot{\cup} \dots \dot{\cup} \mathfrak{C}_{\ell_1} \equiv_{r-1}^M \mathfrak{C}'_2 \dot{\cup} \dots \dot{\cup} \mathfrak{C}'_{\ell_2}$, Duplicator can win the remaining $(r - 1)$ -round game from position (s, d) by Lemma 3.5. ■

Now we have the necessary tools available to prove the main theorem.

Theorem 4.7. $\text{CMSO} \subsetneq \text{MSO}[<]_{\text{inv}}$.

Proof. We show that the class $\mathcal{C} := \{ \mathfrak{G}_{k\ell} : k|\ell \}$ is not definable in CMSO but order-invariantly definable in MSO by the sentence $\psi_{\text{grid}} \wedge \varphi$, where

$$\varphi = \exists \min \exists c \left(\begin{array}{l} \forall x(\min \leq x) \wedge \neg \exists z(E_h(c, z) \vee E_v(c, z)) \\ \wedge \forall T(\forall x \forall y(Tx \wedge \varphi_{\text{diag}}(x, y) \rightarrow Ty) \wedge T \min \rightarrow Tc) \end{array} \right),$$

and

$$\begin{aligned} \varphi_{\text{diag}}(x, y) &= (\exists z(E_v(x, z) \wedge E_h(z, y))) \\ &\quad \vee (\neg \exists z E_v(x, z) \wedge \exists z(z \sim_h \min \wedge z \sim_v x \wedge E_h(z, y))), \\ E_h(x, y) &= x \sim_h y \wedge \exists x_0 \exists y_0 \left(\begin{array}{l} x_0 \sim_h \min \wedge y_0 \sim_h \min \\ \wedge x \sim_v x_0 \wedge y \sim_v y_0 \wedge x_0 < y_0 \\ \wedge \forall z_0(z_0 \sim_h \min \rightarrow z_0 \leq x_0 \vee z_0 \geq y_0) \end{array} \right), \\ E_v(x, y) &= x \sim_v y \wedge \exists x_0 \exists y_0 \left(\begin{array}{l} x_0 \sim_v \min \wedge y_0 \sim_v \min \\ \wedge x \sim_h x_0 \wedge y \sim_h y_0 \wedge x_0 < y_0 \\ \wedge \forall z_0(z_0 \sim_v \min \rightarrow z_0 \leq x_0 \vee z_0 \geq y_0) \end{array} \right). \end{aligned}$$

As hinted above, the horizontal and vertical edge relations (E_h and E_v , respectively) are defined using the successor relation which is induced by an arbitrary ordering on the row (and column) containing the minimal element (\min) which itself serves as the lower left corner of the grid. φ_{diag} defines diagonal steps through the grid that wrap around from the top to the bottom row. Finally, φ states that the pair consisting of the lower left corner (\min) and the upper right corner (c) of the grid is contained in the transitive closure of φ_{diag} . Obviously, there is such a sawtooth-shaped path starting at \min and ending exactly in the upper right corner if, and only if, $k|\ell$.

The second step consists in showing that \mathcal{C} is not definable in CMSO. Towards this goal, we show that for any choice of $r \in \mathbb{N}$ and $M \subseteq_{\text{fin}} \mathbb{N}^+$, we can find $k, \ell_1, \ell_2 \in \mathbb{N}$, such that $\mathfrak{G}_{k\ell_1} \in \mathcal{C}$, $\mathfrak{G}_{k\ell_2} \notin \mathcal{C}$, and $\mathfrak{G}_{k\ell_1} \equiv_r^M \mathfrak{G}_{k\ell_2}$ which contradicts the CMSO-definability of \mathcal{C} .

Let $r \in \mathbb{N}$ and $M \subseteq_{\text{fin}} \mathbb{N}^+$ be fixed. We choose $s \geq r + 1$ such that $2^s \nmid \text{lcm}(M)$. Let $k = 2^s$, $\ell_1 = 2^{kr+1} \text{lcm}(M)$, and $\ell_2 = \ell_1 + \text{lcm}(M)$. Obviously, ℓ_1 and ℓ_2 satisfy the conditions of Lemma 4.6, and thus $\mathfrak{G}_{k\ell_1} \equiv_r^M \mathfrak{G}_{k\ell_2}$.

Furthermore, $\ell_1 = k \cdot 2^{2^s \cdot r - s + 1} \text{lcm}(M)$, hence $k \mid \ell_1$ and $\mathfrak{G}_{k\ell_1} \in \mathcal{C}$. On the other hand, $k \nmid \ell_2 = \ell_1 + \text{lcm}(M)$ by the choice of s , thus $\mathfrak{G}_{k\ell_2} \notin \mathcal{C}$. ■

5. Conclusion

We have provided a characterisation of the expressiveness of CMSO in terms of an Ehrenfeucht-Fraïssé game that naturally extends the known game capturing MSO-definability, and we have presented a class of structures that are shown, using the proposed game characterisation, to be undefinable by a CMSO-sentence yet being definable by an order-invariant MSO-sentence. This establishes that order-invariant MSO is strictly more expressive than counting MSO in the finite. Modifying the separating example by considering a variant of cliquey grids where the two separate equivalence relations are unified into a single binary relation and considering, e.g., the class of such grids where the horizontal dimension exactly matches the vertical dimension, we can also confirm Courcelle's original conjecture.

Corollary 5.1. *CMSO-definability is strictly weaker than $\text{MSO}[<]_{\text{inv}}$ -definability for general graphs.*

The separating query being essentially a transitive closure query, i.e., the only place where monadic second-order quantification is used is in the definition of the transitive closure of a binary relation, we can conclude that the same class of structures yields a separation of $(D)\text{TC}^1[<]_{\text{inv}}$ from $(D)\text{TC}^1$ (the extension of FO by a (deterministic) transitive closure operator on binary relations) and even from $(D)\text{TC}^1$ extended with modulo-counting predicates since $(D)\text{TC}^1 \subseteq \text{MSO}$. Finding separating examples concerning higher arity $(D)\text{TC}$ or even full $(D)\text{TC}$ requires further investigation since, in general, $\text{MSO} \subsetneq \text{DTC}^2$.

Following an opposite line of research, it would be interesting to identify further classes of graphs, besides classes of graphs of bounded tree-width, on which $\text{MSO}[<]_{\text{inv}}$ is no more expressive than CMSO.

References

- [BS05] Michael Benedikt and Luc Segoufin. Towards a characterization of order-invariant queries over tame structures. In *Proceedings of the 14th Annual Conference on Computer Science Logic, CSL 2005*, pages 276–291, 2005.
- [Cou90] Bruno Courcelle. The monadic second-order logic of graphs I: Recognizable sets of finite graphs. *Inform. and Comput.*, 85(1):12–75, 1990.
- [Cou96] Bruno Courcelle. The monadic second-order logic of graphs X: Linear orderings. *Theoretical Computer Science*, 160:87–143, 1996.
- [EF95] Heinz-Dieter Ebbinghaus and Jörg Flum. *Finite Model Theory*. Perspectives in Mathematical Logic. Springer-Verlag, Berlin, 1995.
- [Lap98] Denis Lepoivre. Recognizability equals monadic second-order definability for sets of graphs of bounded tree-width. In *Proceedings of the 15th Annual Symposium on Theoretical Aspects of Computer Science, STACS 1998*, pages 618–628, 1998.
- [Lib04] Leonid Libkin. *Elements of Finite Model Theory*. Springer, 2004.
- [Nur00] Juha Nurmonen. Counting modulo quantifiers on finite structures. *Information and Computation*, 160(1-2):62–87, 2000.
- [Ott95] Martin Otto. A note on the number of monadic quantifiers in monadic Σ_1^1 . *Information Processing Letters*, 53(6):337–339, March 1995.
- [Ott00] Martin Otto. Epsilon-logic is more expressive than first-order logic over finite structures. *Journal of Symbolic Logic*, 65(4):1749–1757, 2000.

CARDINALITY AND COUNTING QUANTIFIERS ON OMEGA-AUTOMATIC STRUCTURES

L. KAISER¹, S. RUBIN², AND V. BÁRÁNY¹

¹ Mathematische Grundlagen der Informatik
RWTH Aachen
E-mail address: {kaiser,vbarany}@informatik.rwth-aachen.de

² Department of Computer Science
University of Auckland
E-mail address: rubin@cs.auckland.ac.nz

ABSTRACT. We investigate structures that can be represented by omega-automata, so called omega-automatic structures, and prove that relations defined over such structures in first-order logic expanded by the first-order quantifiers ‘there exist at most \aleph_0 many’, ‘there exist finitely many’ and ‘there exist k modulo m many’ are omega-regular. The proof identifies certain algebraic properties of omega-semigroups.

As a consequence an omega-regular equivalence relation of countable index has an omega-regular set of representatives. This implies Blumensath’s conjecture that a countable structure with an ω -automatic presentation can be represented using automata on finite words. This also complements a very recent result of Hjörth, Khoussainov, Montalban and Nies showing that there is an omega-automatic structure which has no injective presentation.

1. Introduction

Automatic structures were introduced in [5] and later again in [6, 2] along the lines of the Büchi-Rabin equivalence of automata and monadic second-order logic. The idea is to encode elements of a structure \mathfrak{A} via words or labelled trees (the codes need not be unique) and to represent the relations of \mathfrak{A} via synchronised automata. This way we reduce the first-order theory of \mathfrak{A} to the monadic second-order theory of one or two successors. In particular, the encoding of relations defined in \mathfrak{A} by first order formulas are also regular, and automata for them can be computed from the original automata. Thus we have the fundamental fact that the first-order theory of an automatic structure is decidable.

Depending on the type of elements encoding the structure, the following natural classes of structures appear: automatic (finite words), ω -automatic (infinite words), tree-automatic (finite trees), and ω -tree automatic (infinite trees). Besides the obvious inclusions, for instance that automatic structures are also ω -automatic, there are still some outstanding problems. For instance, a presentation over finite words or over finite trees can be transformed into one where each element has a unique representative.

Key words and phrases: ω -automatic presentations, ω -semigroups, ω -automata.

Kuske and Lohrey [9] point out an ω -regular equivalence relation (namely \sim_e stating that two infinite words are position-wise eventually equal) with no ω -regular set of representatives. Thus, unlike the finite-word case, injectivity can not generally be achieved by selecting a regular set of representatives from a given presentation. In fact, using topological methods it has recently been shown [4] that there are omega-automatic structures having no injective presentation. However, we are able to prove that every omega-regular equivalence relation having only countably many classes does allow to select an omega-regular set of unique representants. Therefore, every countable omega-automatic structure does have an injective presentation.

A related question raised by Blumensath [1] is whether every countable ω -automatic structure is also automatic. In Corollary 2.8 we confirm this by transforming the given presentation into an injective one, and then noting that an injective ω -automatic presentation of a countable structure can be “packed” into one over finite words.

All these results rest on our main contribution: a characterisation of when there exist countably many words x satisfying a given formula with parameters in a given ω -automatic structure \mathfrak{A} (with no restriction on the cardinality of the domain of \mathfrak{A} or the injectivity of the presentation). The characterisation is first-order expressible in an ω -automatic presentation of an extension of \mathfrak{A} by \sim_e . Hence we obtain an extension of the fundamental fact for ω -automatic structures to include cardinality and counting quantifiers such as ‘there exists (un)countably many’, ‘there exists finitely many’, and ‘there exists k modulo m many’. This generalises results of Kuske and Lohrey [9] who achieve this for structures with *injective* ω -automatic presentations.

2. Preliminaries

By countable we mean finite or countably infinite. Let Σ be a finite alphabet. With Σ^* and Σ^ω we denote the set of finite, respectively ω -words over Σ . The length of a word $w \in \Sigma^*$ is denoted by $|w|$, the empty word by ε , and for each $0 \leq i < |w|$ the i th symbol of w is written as $w[i]$. Similarly $w[n, m]$ is the factor $w[n]w[n+1]\cdots w[m]$ and $w[n, m)$ is defined by $w[n, m-1]$. Note that we start indexing with 0 and that for $u \in \Sigma^*$ we denote by u^n the concatenation of n number of u s, in particular $u^\omega \in \Sigma^\omega$.

We consider relations on finite and ω -words recognised by multi-tape finite automata operating in a synchronised letter-to-letter fashion. Formally, ω -regular relations are those accepted by some finite non-deterministic automaton \mathcal{A} with Büchi, parity or Muller acceptance conditions, collectively known as ω -automata, and having transitions labelled by m -tuples of symbols of Σ . Equivalently, \mathcal{A} is a usual one-tape ω -automaton over the alphabet Σ^m accepting the convolution $\otimes\vec{w}$ of ω -words w_1, \dots, w_m defined by $\otimes\vec{w}[i] = (w_1[i], \dots, w_m[i])$ for all i .

Words $u, v \in \Sigma^\omega$ have *equal ends*, written $u \sim_e v$, if for almost all $n \in \mathbb{N}$, $u[n] = v[n]$. This is an important ω -regular equivalence relation. We overload notation so that for $S, T \subset \mathbb{N}$ we write $S \sim_e T$ to mean for almost all $n \in \mathbb{N}$, $n \in S \iff n \in T$.

Example 2.1. The non-deterministic Büchi automaton depicted in Fig. 1 accepts the equal-ends relation on alphabet $\{0, 1\}$.

In the case of finite words one needs to introduce a padding end-of-word symbol $\square \notin \Sigma$ to formally define convolution of words of different length. For simplicity, we shall identify each finite word $w \in \Sigma^*$ with its infinite padding $w^\square = w\square^\omega \in \Sigma_\square^\omega$ where $\Sigma_\square = \Sigma \cup \{\square\}$.

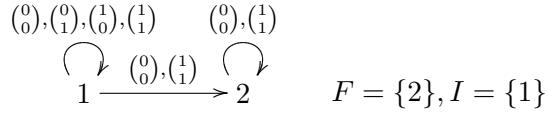


Figure 1: An automaton for the equal ends relation \sim_e .

To avoid repeating the definition of automata for finite words, we say that a m -ary relation $R \subseteq (\Sigma^*)^m$ is *regular* (*synchronised rational*) whenever it is ω -regular over Σ^\square .

2.1. Automatic structures

We now define what it means for a relational structure (we implicitly replace any structure with its relational counterpart) to have an (ω -)automatic presentation.

Definition 2.2 ((ω -)Automatic presentations).

Consider a relational structure $\mathfrak{A} = (A, \{R_i\}_i)$ with universe $\text{dom}(\mathfrak{A}) = A$ and relations R_i . A tuple of ω -automata $\mathfrak{d} = (\mathcal{A}, \mathcal{A}_\approx, \{\mathcal{A}_i\}_i)$ together with a surjective naming function $f : L(\mathcal{A}) \rightarrow A$ constitutes an (ω -)automatic presentation of \mathfrak{A} if the following criteria are met:

- (i) the equivalence, denoted \approx , and defined by $\{(u, w) \in L(\mathcal{A})^2 \mid f(u) = f(w)\}$ is recognised by \mathcal{A}_\approx ,
- (ii) every $L(\mathcal{A}_i)$ has the same arity as R_i ,
- (iii) f is an isomorphism between $\mathfrak{A}_\mathfrak{d} = (L(\mathcal{A}), \{L(\mathcal{A}_i)\}_i)/_\approx$ and \mathfrak{A} .

The presentation is said to be *injective* whenever f is, in which case \mathcal{A}_\approx can be omitted.

The relation \approx needs to be a congruence of the structure $(L(\mathcal{A}), \{L(\mathcal{A}_i)\}_i)$ for item (iii) to make sense. In case $L(\mathcal{A})$ only consists of words of the form w^\square where $w \in \Sigma^*$, we say that the presentation is *automatic*. Call a structure (ω -)automatic if it has an (ω -)automatic presentation.

The advantage of having an (ω -)automatic presentation of a structure lies in the fact that first-order (FO) formulas can be effectively evaluated using classical automata constructions. This is expressed by the following fundamental theorem.

Theorem 2.3. (Cf. [5], [6], [3].)

- (i) There is an effective procedure that given an (ω -)automatic presentation \mathfrak{d}, f of a structure \mathfrak{A} , and given a FO-formula $\varphi(\vec{a}, \vec{x})$ with parameters \vec{a} from \mathfrak{A} (defining a k -ary relation R over \mathfrak{A}), constructs a k -tape synchronous (ω -)automaton recognising $f^{-1}(R)$.
- (ii) The FO-theory of every (ω -)automatic structure is decidable.
- (iii) The class of (ω -)automatic structures is closed under FO-interpretations

Let FOC denote the extension of first-order logic with all quantifiers of the form

- $\exists^{(r \bmod m)} x . \varphi$ meaning that the number of x satisfying φ is finite and is congruent to $r \bmod m$;
- $\exists^\infty x . \varphi$ meaning that there are infinitely many x satisfying φ ;
- $\exists^{\leq \aleph_0} x . \varphi$ and $\exists^{> \aleph_0} x . \varphi$ meaning that the cardinality of the set of all x satisfying φ is countable, or uncountable, respectively.

It has been observed that for *injective* (ω -)automatic presentations Theorem 2.3 can be extended from FO to FOC [8, 9]. Moreover, Kuske and Lohrey show that the cardinality of any set definable in FOC is either countable or equal to that of the continuum. Our main contribution is the following generalisation of their result.

Theorem 2.4. *The statements of Theorem 2.3 hold true for FOC over all (not necessarily injective) ω -automatic presentations.*

It is easily seen that finite-word automatic presentations can be assumed to be injective. This is achieved by restricting the domain of the presentation to a regular set of representatives of the equivalence involved. This can be done effectively, e.g. by selecting the length-lexicographically least word of every class.

This brings us to the question which ω -automatic structures allow an injective ω -automatic presentation. In [9] Kuske and Lohrey have pointed out that not every ω -regular equivalence has an ω -regular set of representatives. In particular, the following Lemma shows that the *equal-ends relation* \sim_e of Example 2.1 is a counterexample.

Lemma 2.5 ([9, Lemma 2.4]). *Let \mathcal{A} be a Büchi automaton with n states over $\Sigma \times \Gamma$ and let $u \in \Sigma^\omega$ be given. Consider the set $V = \{v \in \Gamma^\omega \mid u \otimes v \in L(\mathcal{A})\}$. Then V is uncountable if and only if $|V/\sim_e| > n$, otherwise it is finite or countable.*

The lemma implies that an ω -regular set is countable if and only if it meets only finitely many equal-ends-classes. In this case each of its members is ultimately periodic with one of finitely many periods.

Corollary 2.6. *An ω -regular set is countable iff it can be written as a finite union of sets of the form $U_j \cdot (w_j)^\omega$ with each U_j a regular set of finite words and each w_j a finite word.*

A related question raised by Blumensath [1] is whether every *countable* ω -automatic structure is also automatic. It is easy to see that every *injective* ω -automatic presentation of a countable structure can be “packed” into an automatic presentation.

Proposition 2.7. ([1, Theorem 5.32]) *Let \mathfrak{d} be an injective ω -automatic presentation of a countable structure \mathcal{A} . Then, an (injective) automatic presentation \mathfrak{d}' of \mathcal{A} can be effectively constructed.*

In our proof of Theorem 2.4 we identify a property of finite semigroups that recognise transitive relations (Lemma 3.3 item (3)) that allows us to drop the assumption of injectivity in the previous statement. We are thus able to answer the question of Blumensath.

Corollary 2.8. *A countable structure is ω -automatic if and only if it is automatic. Transforming a presentation of one type into the other can be done effectively.*

2.2. ω -Semigroups

The fundamental correspondence between recognisability by finite automata and by finite semigroups has been extended to ω -regular sets. This is based on the notion of ω -semigroups. Rudimentary facts on ω -semigroups are well presented in [10]. We only mention what is most necessary.

An ω -semigroup $S = (S_f, S_\omega, \cdot, *, \pi)$ is a two-sorted algebra, where (S_f, \cdot) is a semigroup, $* : S_f \times S_\omega \mapsto S_\omega$ is the *mixed product* satisfying for every $s, t \in S_f$ and every $\alpha \in S_\omega$ the equality

$$s \cdot (t * \alpha) = (s \cdot t) * \alpha$$

and where $\pi : S_f^\omega \mapsto S_\omega$ is the *infinite product* satisfying

$$s_0 \cdot \pi(s_1, s_2, \dots) = \pi(s_0, s_1, s_2, \dots)$$

as well as the associativity rule

$$\pi(s_0, s_1, s_2, \dots) = \pi(s_0 s_1 \cdots s_{k_1}, s_{k_1+1} s_{k_1+2} \cdots s_{k_2}, \dots)$$

for every sequence $(s_i)_{i \geq 0}$ of elements of S_f and every strictly increasing sequence $(k_i)_{i \geq 0}$ of indices. For $s \in S_f$ we denote $s^\omega = \pi(s, s, \dots)$.

Morphisms of ω -semigroups are defined to preserve all three products as expected. There is a natural way to extend finite semigroups and their morphisms to ω -semigroups. As in semigroup theory, idempotents play a central role in this extension. An *idempotent* is a semigroup element $e \in S$ satisfying $ee = e$. For every element s in a finite semigroup the sub-semigroup generated by s contains a unique idempotent s^k . The least $k > 0$ such that s^k is idempotent for every $s \in S_f$ is called the *exponent* of the semigroup S_f and is denoted by π . Another useful notion is absorption of semigroup elements: say that s *absorbs* t (*on the right*) if $st = s$.

There is also a natural extension of the free semigroup Σ^+ to the ω -semigroup $(\Sigma^+, \Sigma^\omega)$ with $*$ and π determined by concatenation. An ω -semigroup $S = (S_f, S_\omega)$ *recognises* a language $L \subseteq \Sigma^\omega$ via a morphism $\phi : (\Sigma^+, \Sigma^\omega) \rightarrow (S_f, S_\omega)$ if $\phi^{-1}(\phi(L)) = L$. This notion of recognisability coincides, as for finite words, with that by non-deterministic Büchi automata. In [10] constructions from Büchi automata to ω -semigroups and back are also presented.

Theorem 2.9 ([10]).

A language $L \subseteq \Sigma^\omega$ is ω -regular iff it is recognised by a finite ω -semigroup.

We note that this correspondence allows one to engage in an algebraic study of varieties of ω -regular languages, and also has the advantage of hiding complications of cutting apart and stitching together runs of Büchi automata as we shall do. This is precisely the reason that we use this algebraic framework. Most remarkably, one does not need to understand the exact relationship between automata and ω -semigroups and the technical details of the constructions behind Theorem 2.9 to comprehend our proof. An alternative approach, though likely less advantageous, would be to use the composition method, which is closer in spirit to ω -semigroups than to automata.¹

3. Cardinality and modulo counting quantifiers

This section is devoted to establishing the key to Theorem 2.4 announced earlier.

We characterise when there exist countably many words x satisfying a given formula with parameters $\varphi(x, \vec{z})$ in some ω -automatic structure \mathfrak{A} . The characterisation is first-order expressible in an ω -automatic extension of \mathfrak{A} by the equal-ends relation \sim_e .

So, fix an ω -automatic presentation of some \mathfrak{A} with congruence \approx , and a first-order formula $\varphi(x, \vec{z})$ in the language of \mathfrak{A} with x and \vec{z} free variables.

Proposition 3.1. *There is a constant C , computable from the presentation \mathfrak{d} , so that for all tuples \vec{z} of infinite words the following are equivalent:*

- (1) $\varphi(-, \vec{z})$ is satisfiable and \approx restricted to the domain $\varphi(-, \vec{z})$ has countably many equivalence classes.

¹Define T_f resp. T_ω as the sets of bounded (in terms of quantifier rank) theories of finite, respectively, of ω -words. The composition theorem ensures that $\cdot, *, \pi$ can naturally be defined on bounded theories.

- (2) there exist C -many words x_1, \dots, x_C each satisfying $\varphi(-, \vec{z})$, so that every x satisfying $\varphi(-, \vec{z})$ is \approx -equivalent to some $y \sim_e x_i$. Formally, the structure $(\mathfrak{A}, \approx, \sim_e)$ models the sentence below.

$$\forall \vec{z} \left(\exists^{\leq \aleph_0} w . \varphi(w, \vec{z}) \longleftrightarrow \exists x_1 \dots x_C \left(\bigwedge_i \varphi(x_i, \vec{z}) \wedge \forall x \varphi(x, \vec{z}) \rightarrow \exists y (x \approx y \wedge \bigvee_i y \sim_e x_i) \right) \right)$$

Proof. Suppose \mathfrak{d} , \mathfrak{A} , and φ are given. Define C to be c^2 , where c is the size of the largest ω -semigroup corresponding to any of the given automata (from the presentation or corresponding to φ). Now fix parameters \vec{z} . From now on, \approx denotes the equivalence relation \approx restricted to domain $\varphi(-, \vec{z})$.

2 → 1: Condition 2 and the fact that every \sim_e -class is countable imply that all words satisfying $\varphi(-, \vec{z})$ are contained in a countable number of \approx -classes.

1 → 2: We prove the contra-positive in three steps.

If $\varphi(-, \vec{z})$ is satisfiable then the negation of condition 2 implies that there are $C + 1$ many words x_0, \dots, x_C each satisfying $\varphi(-, \vec{z})$, and so that for $i, j \leq C$, $i \neq j$, the \approx -class of x_j does not meet the \sim_e -class of x_i . In particular, the x_i s are pairwise $\not\sim_e$.

The plan is to produce uncountably many pairwise non- \approx words that satisfy $\varphi(-, \vec{z})$. In the first 'Ramsey step', similar to what is done in [9], we find two words from the given C many, say $x_1, x_2 \in \Sigma^*$, and a factorisation $H \subset \mathbb{N}$ so that both words behave the same way along the factored sub-words with respect to the \approx - and φ -semigroups. In the second 'Coarsening step' we identify a technical property of finite semigroups recognising transitive relations. This allows us to produce an altered factorisation G and new, well-behaving words y_1, y_2 . In the final step, the new words are 'shuffled along G ' to produce continuum many pairwise non- \approx words, each satisfying $\varphi(-, \vec{z})$.

3.1. Ramsey step

This step effectively allows us to discard the parameters \vec{z} . Before we use Ramsey's theorem, we introduce a convenient notation to talk about factorisations of words.

Definition 3.2. Let $A = a_1 < a_2 < \dots$ be any subset of \mathbb{N} and $h : \Sigma^* \rightarrow S$ be a morphism into a finite semigroup S . For an ω -word $\alpha \in \Sigma^\omega$, and element $e \in S$, say that A is an h, e -homogeneous factorisation of α if for all $n \in \mathbb{N}^+$, $h(\alpha[a_n, a_{n+1}]) = e$.

Observe that

- (1) if A is an h, s -homogeneous factorisation of α and $k \in \mathbb{N}^+$ then the set $\{a_{ki}\}_{i \in \mathbb{N}^+}$ is an h, s^k -homogeneous factorisation of α .
- (2) if A is an h, e -homogeneous factorisation of α and e is idempotent, then every infinite $B \subset A$ is also an h, e -homogeneous factorisation of α .

In the following we write w^φ and w^\approx to denote the image of w under the semigroup morphism into the finite semigroup associated to φ and \approx , respectively, as determined by the presentation. Accordingly, we will speak of e.g. φ, s_i -homogeneous factorisations.

Let us now colour every $\{n, m\} \in [\mathbb{N}]^2$, say $n < m$, by the tuple of ω -semigroup elements

$$\langle (\otimes (x_i, \vec{z})[n, m])^\varphi)_{0 \leq i \leq C}, (\otimes (x_i, x_j)[n, m])^\approx)_{0 \leq i \leq j \leq C} \rangle.$$

By Ramsey's theorem there exists infinite $H \subset \mathbb{N}$ and a tuple of ω -semigroup elements

$$\langle (s_i)_{1 \leq i \leq C}, (t_{(i,j)})_{1 \leq i \leq j \leq C} \rangle$$

so that for all $0 \leq i \leq j \leq C$,

- H is a φ, s_i -homogeneous factorisation of the word $\otimes(x_i, \vec{z})$,
- H is $\approx, t_{(i,j)}$ -homogeneous factorisation of the word $\otimes(x_i, x_j)$.

Note that by virtue of additivity of our colouring and Ramsey's theorem each of the s_i and $t_{(i,j)}$ above are idempotents. Note that since there are at most c -many s_i s and c -many $t_{(i,i)}$ s there are at most c^2 many pairs $(s_i, t_{(i,i)})$ and so there must be two indices, we may suppose 1 and 2, with $s_1 = s_2$ and $t_{(1,1)} = t_{(2,2)}$.

3.2. Coarsening step

For technical reasons we now refine H and alter x_1, x_2 so that the semigroup elements have certain additional properties.

To start with, using the fact that $x_1 \not\sim_e x_2$ and our observation on coarsenings, we assume without loss of generality that H is coarse enough so that $x_1[h_n, h_{n+1}] \neq x_2[h_n, h_{n+1}]$ for all $n \in \mathbb{N}$.

Lemma 3.3. *There exists a subset $G \subset H$, listed as $g_1 < g_2 < \dots$, and ω -words y_1, y_2 with the following properties:*

- (1) *The words y_1 and y_2 are neither \approx -equivalent nor \sim_e -equivalent, and each satisfies $\varphi(-, \vec{z})$.*
- (2) *There exists an idempotent φ -semigroup element s such that G is a φ, s -homogeneous factorisation for each of $\otimes(y_1, \vec{z})$ and $\otimes(y_2, \vec{z})$.*
- (3) *There exist idempotent \approx -semigroup elements $t, t^\uparrow, t^\downarrow$ so that for $y_j \in \{y_1, y_2\}$*
 - *both t^\uparrow and t^\downarrow absorb t*
 - *$\otimes(y_j, y_j)[0, g_1] \approx$ absorbs t*
 - *G is an \approx, t -homogeneous factorisation of $\otimes(y_j, y_j)$*
 - *G is an \approx, t^\uparrow -homogeneous factorisation of $\otimes(y_1, y_2)$*
 - *G is an \approx, t^\downarrow -homogeneous factorisation of $\otimes(y_2, y_1)$.*

Proof. Define ω -words $y_1 := x_2[0, h_2)x_1[h_2, \infty)$, and y_2 by

$$\begin{aligned} y_2[0, h_2) &:= x_2[0, h_2) \text{ and} \\ y_2[h_{2n}, h_{2n+2}) &:= x_2[h_{2n}, h_{2n+1})x_1[h_{2n+1}, h_{2n+2}) \text{ for } n > 0. \end{aligned}$$

Item 1. Clearly, $y_1 \not\sim_e y_2$ and each $y_j \in \{y_1, y_2\}$ satisfies $\varphi(y_j, \vec{z})$ since by homogeneity and $s_1 = s_2$

$$\begin{aligned} \otimes(y_1, \vec{z})^\varphi &= \otimes(x_2, \vec{z})[0, h_2)^\varphi s_1^\omega \\ &= \otimes(x_2, \vec{z})[0, h_2)^\varphi s_2^\omega \\ &= \otimes(x_2, \vec{z})^\varphi \end{aligned}$$

and similarly

$$\begin{aligned} \otimes(y_2, \vec{z})^\varphi &= \otimes(x_2, \vec{z})[0, h_2)^\varphi (s_2 s_1)^\omega \\ &= \otimes(x_2, \vec{z})[0, h_2)^\varphi s_2^\omega \\ &= \otimes(x_2, \vec{z})^\varphi \end{aligned}$$

Next we check that $y_1 \not\approx y_2$.

$$\begin{aligned}
\otimes(y_1, y_2)^\approx &= \pi_\approx(\otimes(x_2, x_2)[0, h_2]^\approx, (\otimes(x_1, x_2)[h_{2n}, h_{2n+1}]^\approx, \otimes(x_1, x_1)[h_{2n+1}, h_{2n+2}]^\approx)_{n \in \mathbb{N}^+}) \\
&= \otimes(x_2, x_2)[0, h_1]^\approx t_{(2,2)} (t_{(1,2)} t_{(1,1)})^\omega \\
&= \otimes(x_2, x_2)[0, h_1]^\approx t_{(2,2)} t_{(2,2)} (t_{(1,2)} t_{(1,1)})^\omega \\
&= \otimes(x_2, x_2)[0, h_1]^\approx t_{(2,2)} t_{(2,2)} (t_{(1,2)} t_{(2,2)})^\omega \\
&= \otimes(x_2, x_2)[0, h_1]^\approx t_{(2,2)} (t_{(2,2)} t_{(1,2)})^\omega \\
&= \pi_\approx(\otimes(x_2, x_2)[0, h_2]^\approx, (\otimes(x_2, x_2)[h_{2n}, h_{2n+1}]^\approx, \otimes(x_1, x_2)[h_{2n+1}, h_{2n+2}]^\approx)_{n \in \mathbb{N}^+}) \\
&= \otimes(y_2, x_2)^\approx
\end{aligned}$$

Thus, if $y_1 \approx y_2$ then also $y_2 \approx x_2$ and so by **transitivity** $y_1 \approx x_2$. But since $y_1 \sim_e x_1$, the \approx -class of x_2 meets the \sim_e -class of x_1 , contradicting the initial choice of the x_i s.

Items 2 and 3. Define intermediate semigroup elements $q := s_1$, $r := t_{(1,1)}$, $r^\uparrow := t_{(1,2)} t_{(1,1)}$ and $r^\downarrow := t_{(2,1)} t_{(1,1)}$. Then

- (1) both r^\uparrow and r^\downarrow absorb r , since $t_{(1,1)}$ is idempotent;
- (2) $\otimes(y_j, y_j)[0, h_2]^\approx = \otimes(y_j, y_j)[0, h_1]^\approx t_{(2,2)}$ and thus absorbs r (for $y_j \in \{y_1, y_2\}$).

In this notation, for all $i \in \mathbb{N}^+$ and $y_j \in \{y_1, y_2\}$,

- $\otimes(y_j, \vec{z})[h_{2i}, h_{2i+2}]^\varphi$ is $qq = q$,
- $\otimes(y_j, y_j)[h_{2i}, h_{2i+2}]^\approx$ is $rr = r$,
- $\otimes(y_1, y_2)[h_{2i}, h_{2i+2}]^\approx$ is $t_{(1,2)} t_{(1,1)} = r^\uparrow$,
- $\otimes(y_2, y_1)[h_{2i}, h_{2i+2}]^\approx$ is $t_{(2,1)} t_{(1,1)} = r^\downarrow$.

Finally, define the set $G := \{h_{2ki}\}_{i>1}$, i.e. $g_i = h_{2k(i+1)}$, and the semigroup elements $t := r^k$, $t^\uparrow := (r^\uparrow)^k$, $t^\downarrow := (r^\downarrow)^k$ and $s := q^k$. The extra multiple of k (defined as the product of the exponents of the give semigroups for \sim_e and \approx) ensures all these semigroup elements (in particular t^\uparrow and t^\downarrow) are idempotent. We now verify the absorption properties:

$$t^\uparrow t = r^{\uparrow k} r^k = r^{\uparrow k} = t^\uparrow \quad \text{because } r^\uparrow \text{ absorbs } r$$

Similarly, $t^\downarrow t$ absorbs t . Further, since $g_1 = h_{4k}$, we have

$$\begin{aligned}
\otimes(y_j, y_j)[0, g_1]^\approx &= \otimes(y_j, y_j)[0, h_2]^\approx \otimes(y_j, y_j)[h_2, h_{4k}]^\approx \\
&= \otimes(y_j, y_j)[0, h_2]^\approx r^{4k-2} \\
&= \otimes(y_j, y_j)[0, h_2]^\approx r^{3k-2} t
\end{aligned}$$

and thus absorbs t .

Finally we verify the homogeneity properties: G is an \approx, t^\downarrow -homogeneous factorisation of $\otimes(y_2, y_1)$ since for $i \in \mathbb{N}^+$

$$\otimes(y_2, y_1)[g_i, g_{i+1}]^\approx = \otimes(y_2, y_1)[h_{2k(i+1)}, h_{2k(i+2)}]^\approx = (r^\downarrow)^k = t^\downarrow.$$

The other cases are similar. ■

3.3. Shuffling step

We continue the proof of Proposition 3.1 by 'shuffling' the words y_1 and y_2 along G resulting in continuum many pairwise distinct words that are pairwise not \approx -equivalent, each satisfying $\varphi(-, \vec{z})$. To this end, define for $S \subset \mathbb{N}^+$ the 'characteristic word' χ_S by

$$\begin{aligned}\chi_S[0, g_1) &:= y_2[0, g_1), \text{ and} \\ \chi_S[g_n, g_{n+1}) &:= \begin{cases} y_2[g_n, g_{n+1}) & \text{if } n \in S, \\ y_1[g_n, g_{n+1}) & \text{otherwise.} \end{cases}\end{aligned}$$

First note that $\mathfrak{A} \models \varphi(\chi_S, \vec{z})$. Indeed, by Lemma 3.3 item 2

$$\begin{aligned}\otimes(\chi_S, \vec{z})^\varphi &= \otimes(y_2, \vec{z})[0, g_1)^\varphi s^\omega \\ &= \otimes(y_2, \vec{z})^\varphi\end{aligned}$$

and $\mathfrak{A} \models \varphi(y_2, \vec{z})$ by Lemma 3.3 item 1. Moreover, for $S \not\sim_e T$ the construction gives that $\chi_S \not\sim_e \chi_T$. This is due our initial choice of $x_1 \not\sim_e x_2$ and the assumption that the factorisation $(h_n)_n$ is coarse enough so that $x_1[h_n, h_{n+1}] \neq x_2[h_n, h_{n+1}]$ and therefore also $y_1[g_n, g_{n+1}] \neq y_2[g_n, g_{n+1}]$ for all n .

The following two lemmas establish that if both $S \setminus T$ and $T \setminus S$ are infinite then $\chi_S \not\approx \chi_T$. Write $x_{\bullet\bullet}$ for the word $\chi_{2\mathbb{N}^+}$, and $x_{\bullet\circ}$ for $\chi_{2\mathbb{N}^+-1}$ and let p denote $\otimes(y_2, y_2)[0, g_1]^\approx$.

Lemma 3.4. *For all S, T with both $S \setminus T$ and $T \setminus S$ infinite,*

$$\otimes(\chi_S, \chi_T)^\approx = \begin{cases} \otimes(x_{\bullet\bullet}, x_{\bullet\bullet})^\approx & \text{or} \\ \otimes(x_{\bullet\circ}, x_{\bullet\circ})^\approx. \end{cases}$$

Proof. Define semigroup-elements p_n for $n \in \mathbb{N}$ by

$$p_n := \begin{cases} t^\downarrow & \text{if } n \in S \setminus T, \\ t^\uparrow & \text{if } n \in T \setminus S, \\ t & \text{otherwise.} \end{cases}$$

Let m be the smallest number in $S \Delta T$. Suppose that $m \in S \setminus T$. Because both t^\uparrow and t^\downarrow are idempotent and since t is absorbed by both p , t^\uparrow and t^\downarrow we have

$$\begin{aligned}\otimes(\chi_S, \chi_T)^\approx &= \pi_\approx(p, (p_n)_{n \in \mathbb{N}}) = p(t^\downarrow t^\uparrow)^\omega \\ &= \otimes(x_{\bullet\circ}, x_{\bullet\circ})^\approx\end{aligned}$$

and the case that $m \in T \setminus S$ similarly results in $\otimes(x_{\circ\bullet}, x_{\circ\bullet})^\approx$. ■

Lemma 3.5. $x_{\circ\bullet} \not\approx x_{\bullet\circ}$.

Proof. Define an intermediate word $x_{\circ\bullet\circ\circ} := \chi_{4\mathbb{N}^+-2}$. By computations similar to the above we find that

$$\begin{aligned}\otimes(x_{\bullet\circ}, x_{\circ\bullet\circ\circ})^\approx &= p(t^\downarrow t^\uparrow t^\downarrow t)^\omega = p(t^\downarrow t^\uparrow t^\downarrow)^\omega = p(t^\downarrow t^\uparrow)^\omega \\ &= \otimes(x_{\bullet\circ}, x_{\circ\bullet})^\approx\end{aligned}$$

and

$$\begin{aligned}\otimes(x_{\circ\bullet}, x_{\circ\bullet\circ\circ})^\approx &= p(tttt^\downarrow)^\omega = p(t^\downarrow)^\omega \\ &= \otimes(y_2, y_1)^\approx\end{aligned}$$

Therefore, if $x_{\circ\bullet} \approx x_{\bullet\circ}$ then also $x_{\bullet\circ} \approx x_{\circ\bullet\circ\circ}$ and so **by symmetry and by transitivity** $x_{\circ\bullet} \approx x_{\circ\bullet\circ\circ}$. But in this case also $y_2 \approx y_1$, contradicting Lemma 3.3 item 1. ■

There are continuum many classes in $\mathcal{P}(\mathbb{N})/\sim_e$. If $S' = 2(\mathbb{N} \setminus S) \cup (2S + 1)$ then $S \not\sim_e T$ implies that both $S' \setminus T'$ and $T' \setminus S'$ are infinite. Thus there is a continuum of pairwise not \approx -equivalent words $\chi_{S'}$ each satisfying $\varphi(-, \vec{z})$. This completes the proof of Proposition 3.1. ■

4. Consequences

Theorem 2.4 *The statements of Theorem 2.3 hold true for FOC over all (not necessarily injective) ω -automatic presentations.*

Proof. We prove item (i) from which the rest of the theorem follows immediately. We inductively eliminate occurrences of cardinality and modulo-counting quantifiers in the following way.

The countability quantifier $\exists^{\leq \aleph_0}$ and uncountability quantifier $\exists^{> \aleph_0}$ can be eliminated (in an extension of the presentation by \sim_e) by the formula given in Proposition 3.1.

For the remaining quantifiers we further expand the presentation with the ω -regular relations

- $\pi(a, b, c)$ saying that $a \sim_e b \sim_e c$ and the last position where a differs from c is no larger than the last position where b differs from c , and
- $\lambda(a, b, c)$ saying that $\pi(a, b, c)$ and $\pi(b, a, c)$ and, writing k for this common position, the word $a[0, k]$ is lexicographically smaller than the word $b[0, k]$.

Now $\exists^{<\infty} \varphi(x, \vec{z})$ is equivalent to

$$\exists x_1 \cdots x_C \Psi(x_1, \dots, x_C, \vec{z})$$

where Ψ expresses that x_1, \dots, x_C satisfy $\varphi(-, \vec{z})$ and there exists a position, say $k \in \mathbb{N}$, so that every \approx -class contains a word satisfying $\varphi(-, \vec{z})$ that coincides with one of the x_i from position k onwards. This additional condition can be expressed by

$$\exists y_1 \cdots y_C \forall x \exists y \left(\varphi(x, \vec{z}) \rightarrow x \approx y \wedge \bigvee_i \pi(y, y_i, x_i) \right)$$

Consequently, $\exists^{(r \bmod m)} x \varphi(x, \vec{z})$ can be eliminated since we can pick out unique representatives of the \approx -classes as those x so that, writing $i(w)$ for the smallest index i for which $w \sim_e x_i$, for every $y \neq x$ in the same \approx -class as x , either

- $i(x) < i(y)$, or
- $i(x) = i(y)$ and $\lambda(x, y, x_{i(x)})$.

Now we can apply the construction of [9] or [8] for elimination of the $\exists^{(r \bmod m)}$ quantifier. ■

As a corollary of Proposition 3.1 we obtain that for every omega-regular equivalence with countably many classes a set of unique representants is definable.

Corollary 4.1. *Let \approx be an ω -automatic equivalence relation on Σ^ω . There is a constant C , depending on the presentation, so that the following are equivalent:*

- (1) \approx has countably many equivalence classes.
- (2) there exist C many \sim_e -classes so that every \approx -class has non-empty intersection with at least one of these C .

In this case there is an ω -regular set of representatives of \approx . Moreover an automaton for this set can be effectively found given an automaton for \approx .

Proof. The first two items are simply a specialisation of Proposition 3.1. We get the representatives as follows.

Write A for the domain of \approx and consider the formula $\psi(x_1, \dots, x_C)$ with free variables x_1, \dots, x_C :

$$\bigwedge_i x_i \in A \wedge (\forall x \in A) (\exists y) [x \approx y \wedge \bigvee_i y \sim_e x_i]$$

The relation defined by ψ is ω -regular since it is a first order formula over ω -regular relations. By assumption it is non-empty. Thus it contains an ultimately periodic word of the form $\otimes(a_1, \dots, a_C)$. Thus each of these a_i s is ultimately periodic; say $a_i = v_i(u_i)^\omega$.

Then every x has an \approx -representative in $B := \bigcup_i \Sigma^*(u_i)^\omega$. It remains to prune B to select unique representatives for each \approx -class.

It is easy to construct an ω -regular well-founded linear order on B . For every $w \in B$, let $p(w) \in \Sigma^*$ be the length-lexicographically smallest word such that w has period $p(w)$. Also let $t(w) \in \Sigma^*$ be the length-lexicographically smallest word so that $w = t(w) \cdot p(w)^\omega$. Define an order \prec on B by $w \prec w'$ if $p(w)$ is length-lexicographically smaller than $p(w')$, or otherwise if $p(w) = p(w')$ and $t(w)$ is length-lexicographically smaller than $t(w')$. The ordering \prec is ω -regular since it is FO-definable in terms of ω -regular relations. Finally, the required set of representatives may be defined as the set of \prec -minimal elements of every \approx -class; and an automaton for this set can be constructed from an automaton for \approx . ■

This immediately yields an *injective* ω -automatic presentation from a given ω -automatic presentation which by Proposition 2.7 can be transformed into an automatic presentation of the structure. Thus we conclude that every countable ω -automatic structure is already automatic.

Corollary 2.8 *A countable structure is ω -automatic if and only if it is automatic. Transforming a presentation of one type into the other can be done effectively.*

Note that some of our technical results, in particular Lemmas 3.3 and 3.4, only require transitivity of the relation \approx and do not use symmetry. Applying them to an ω -automatic linear order \prec we get an interesting uncountable set of words of the form χ_S , $S \subseteq \mathbb{N}$. For any two such words with $S \not\sim_e T$, whether $\chi_S \prec \chi_T$ or not depends only on the first position $m \in S \Delta T$. Thus, \prec behaves like the lexicographic order on such words.

4.1. Failure of Löwenheim-Skolem theorem for ω -automatic structures

While so far the area of automatic structures has mainly focused on individual structures, it is interesting to look at their theories as well. We note a consequence of our work for 'automatic model theory'.

An automatic version of the Downward Löwenheim-Skolem Theorem would say that every uncountable ω -automatic structure has a countable elementary substructure that is also ω -automatic. Unfortunately this is false since there is a first-order theory with an ω -automatic model but no countable ω -automatic model. Indeed, consider the first-order theory of atomless Boolean Algebras. Kuske and Lohrey [9] have observed that it has

an uncountable ω -automatic model, namely $(\mathcal{P}(\mathbb{N}), \cap, \cup, \neg)/\sim_e$. However, Khoussainov et al. [7] show that the countable atomless Boolean algebra is not automatic and so, by Corollary 2.8, neither ω -automatic.

Here is the closest we can get to an automatic Downward Löwenheim-Skolem Theorem for ω -automatic structures.

Proposition 4.2. *Let $(D, \approx, \{R_i\}_{i \leq \omega})$ be an omega-automatic presentation of \mathfrak{A} and let \mathfrak{A}_{up} be its restriction to the ultimately periodic words of D . Then \mathfrak{A}_{up} is a countable elementary substructure of \mathfrak{A} .*

Proof. Relying on the Tarski-Vaught criterion for elementary substructures we only need to show that for all first-order formulas $\varphi(\vec{x}, y)$ and elements \vec{b} of \mathfrak{A}_{up}

$$\mathfrak{A} \models \exists y \varphi(\vec{b}, y) \Rightarrow \mathfrak{A}_{up} \models \exists y \varphi(\vec{b}, y).$$

By Theorem 2.3 $\varphi(\vec{x}, y)$ defines an omega-regular relation and, similarly, since the parameters \vec{b} are all ultimately periodic the set defined by $\varphi(\vec{b}, y)$ is omega-regular. Therefore, if it is non-empty, then it also contains an ultimately periodic word, which is precisely what we needed. ■

This proof can be viewed as a model construction akin to a classical compactness proof. Indeed, starting with ultimately constant words and throwing in witnesses for all existential formulas satisfied in \mathfrak{A} in each round one constructs an increasing sequence of substructures comprising ultimately periodic words of increasing period lengths. The union of these is closed under witnesses by construction. The argument is valid for relational structures with constants assuming that every constant is represented by an ultimately periodic word.

Future work It remains to be seen whether statements analogous to Theorem 2.4 and Corollary 2.8 also hold for automatic presentations over infinite trees.

Acknowledgment We thank the referees for detailed technical remarks and corrections.

References

- [1] A. Blumensath. Automatic structures. Diploma thesis, RWTH-Aachen, 1999.
- [2] A. Blumensath and E. Grädel. Automatic Structures. In *Proceedings of 15th IEEE Symposium on Logic in Computer Science LICS 2000*, pages 51–62, 2000.
- [3] A. Blumensath and E. Grädel. Finite presentations of infinite structures: Automata and interpretations. *Theory of Comp. Sys.*, 37:641 – 674, 2004.
- [4] G. Hjörth, B. Khoussainov, A. Montalban, and A. Nies. Borel structures. Manuscript, 2007.
- [5] B.R. Hodgson. Décidabilité par automate fini. *Ann. sc. math. Québec*, 7(1):39–57, 1983.
- [6] B. Khoussainov and A. Nerode. Automatic presentations of structures. In *LCC ’94*, volume 960 of *LNCS*, pages 367–392. Springer-Verlag, 1995.
- [7] B. Khoussainov, A. Nies, S. Rubin, and F. Stephan. Automatic structures: Richness and limitations. In *LICS*, pages 44–53. IEEE Comp. Soc., 2004.
- [8] B. Khoussainov, S. Rubin, and F. Stephan. Definability and regularity in automatic structures. In *STACS ’04*, volume 2996 of *LNCS*, pages 440–451, 2004.
- [9] D. Kuske and M. Lohrey. First-order and counting theories of ω -automatic structures. In *FoSSaCS*, pages 322–336, 2006.
- [10] D. Perrin and J.-E. Pin. Semigroups and automata on infinite words. In J. Fountain, editor, *Semigroups, Formal Languages and Groups*, NATO Advanced Study Institute, pages 49–72. Kluwer academic publishers, 1995.

AUTOMATA PRESENTING STRUCTURES: A SURVEY OF THE FINITE STRING CASE

SASHA RUBIN

Abstract. A structure has a (finite-string) *automatic presentation* if the elements of its domain can be named by finite strings in such a way that the coded domain and the coded atomic operations are recognised by synchronous multitape automata. Consequently, every structure with an automatic presentation has a decidable first-order theory. The problems surveyed here include the classification of classes of structures with automatic presentations, the complexity of the isomorphism problem, and the relationship between definability and recognisability.

CONTENTS

1. Introduction	170
1.1. Summary	172
1.2. Scope and related work	173
1.3. Acknowledgement	174
2. Definitions	174
2.1. Notation	174
2.2. Synchronous finite automata	175
2.3. Automatic presentations	176
2.4. Examples	178
3. Properties	178
3.1. Fundamental properties	178
3.2. Extending first-order	181
3.3. Automatic presentations	188
3.4. Intrinsic regularity	190
3.5. Restriction on growth	191
3.6. Isomorphism problem	195
4. Classifications	197
4.1. Equivalence structures	198
4.2. Linear orders	199

Received February 2, 2004.

© 2008, Association for Symbolic Logic
1079-8986/08/1402-0001/\$5.10

4.3. Trees	202
4.4. Boolean algebras	203
4.5. Groups, rings and fields	205
5. Open problems	206

§1. Introduction. Which infinite structures can be stored and manipulated by a computer? At the very least, the structure should be representable in a finite amount of space. Moreover, operations that one would like to perform on the structure, typically logical queries, should be computable.

Finite automata are a robust model of computation that seem well suited to describing infinite structures. Although automata classically recognise sets of strings, they can be generalised to recognise n -ary relations by introducing n input-tapes. A (*finite-string*) *automatic presentation* of a structure is a coding of its domain as a set of finite strings so that the domain and each of the atomic operations is recognised by an automaton operating synchronously on its inputs (Definition 2.5)

Automatic presentations are a refinement of computable presentations, and have two salient properties: 1) given an automatic presentation of a structure \mathcal{A} , every first-order definable relation in \mathcal{A} , allowing parameters, is computable by a finite automaton (Theorem 3.1); 2) there are automatically presentable structures, such as certain expansions of the standard model of Presburger arithmetic, that interpret every automatically presentable structure (Theorem 3.8).

The effectiveness of using automata to present infinite structures has been amply displayed in the related concept of automatic groups. These are finitely-generated groups whose Cayley graphs (over a certain natural encoding) are recognised by automata. They were introduced by Thurston in 1986 who was motivated by work of Cannon [18] on hyperbolic groups. Automatic groups form a rich collection of finitely presented groups with tractable algorithmic properties that are undecidable in the general case. For instance, in an automatic group the word problem is solvable in quadratic time and the group elements can be efficiently enumerated. Moreover, every automatic group is also finitely presentable (in the group theoretic sense), and a presentation can be extracted from the automata. Software to find and work with automatic groups has been developed, and packaged with the computational algebra tools GAP and MAGMA; see Holt [33]. The standard reference for automatic groups is the book by Cannon, Epstein, Holt, Levy, Paterson, and Thurston [19]. Automatic groups are not covered in this survey.

The connections between automata and logic that are relevant to structures with automatic presentations can be traced to Büchi [16], Elgot [27] and Trahtenbrot [56]. They characterised, in logical terms, the relations over

finite strings recognisable by automata (Theorem 3.8 (iv)). In particular they established the decidability of WS1S: the weak monadic second-order theory of the structure \mathbb{N} with the successor function. This technique—relating definability and automata to solve decision problems for certain theories—was generalised to automata working on infinite strings (known as ω -automata) for the monadic second-order theory of one successor S1S [17], to automata on finite trees for WS2S [54, 25], and to automata on infinite trees for S2S [49]. However, structures presentable by these more general automata have received less study, and so this survey focuses solely on the finite string case.

The importance of these particular theories is that many first-order theories can be interpreted in them. For instance, Büchi proved that the first-order theory of $(\mathbb{N}, +)$ is decidable by showing that it is interpretable in WS1S. Taking this lead, Hodgson [30, 31, 32] called the first-order-theory of a structure ‘automaton decidable’, if after coding elements of the structure as finite or infinite strings, one can effectively construct automata recognising the encodings of its first-order definable relations. He was interested in the connections between two different ways of proving decidability: via automata and via algebraic constructions (such as direct product and direct power). He observed that certain products of (ω -)automatically presentable structures are again (ω -)automatically presentable, and thus decidable.¹

Khoussainov and Nerode [34], independently of Hodgson and inspired by the success of using automata to describe groups as in [19], introduced structures presentable by automata as part of complexity-theoretic model theory (also called feasible model theory). Here is a brief description of this area.

The general idea is to fix a complexity class \mathcal{C} (such as polynomial time, exponential time, etc.) and study algorithmic properties of \mathcal{C} -structures: those that can be represented by machines operating with complexity in \mathcal{C} . A typical question is whether a given infinite structure is isomorphic to a \mathcal{C} -structure. For instance, in the 1980’s Nerode, Remmel, and Cenzer led the development of polynomial-time structures; see the survey [20]. They prove, for instance, that every computable structure is computably isomorphic to a polynomial-time structure over a binary alphabet.

Structures presentable by automata are part of complexity-theoretic model theory—take the complexity class corresponding to synchronous multitape automata.

Khoussainov and Nerode suggested in [34] that the algebraic-, model theoretic-, and complexity theoretic-properties of automatically presentable structures are amenable to systematic investigation. For instance, they characterised the class of automatically presentable structures via a generalisation of the Myhill-Nerode Theorem for regular languages [34, Theorem 3.3].

¹An ω -automatic presentation, also called Büchi-automatic presentation, is similar to an automatic presentation, except that infinite strings rather than finite strings are used to code the domain of the structure.

Blumensath and Grädel [9, 12] introduced automatic presentations of structures to the logic in computer science community. They established many fundamental results, such as the characterisation of automatically presentable structures via interpretability (Theorem 3.8). They discussed the complexity of various problems associated with evaluating formulas, such as model checking various fragments of first-order logic on automatically presentable structures.² They also provided the fundamental results for the classes of structures presentable by ω -automata and (ω) -tree automata.

My co-authors and I have focused on classifying classes of automatically presented structures in terms of classical invariants. For instance, what can be said about the Cantor-normal form of automatically presentable ordinals? (This question was asked by Khoussainov and Nerode [34] and solved by Delhommé [24]). Or what can be said about the Cantor–Bendixson rank of automatically presentable trees or of automatically presentable Boolean algebras? We aim for positive results of the form ‘A structure from a certain class is automatically presentable if and only if it has certain algebraic properties’, and negative results of the form ‘The isomorphism problem for a certain class of automatically presented structures is complete for a certain level of the arithmetic/analytic hierarchy’. These are discussed in detail in Section 4. Since the condition of being automatically presentable is quite strong, it is not surprising that some classes of structures (ordinals and Boolean algebras for instance) are simple, in the sense that they are easy to describe. Surprisingly, some classes turn out to be complex, in the sense that it is hard to detect if two members are isomorphic. Notably, the isomorphism problem for the class of all automatically presented structures, easily seen to be undecidable, is Σ_1^1 -complete (Theorem 3.53). Finally, there are many classes (groups, rings, and linear orders for instance) for which it is not yet known whether their automatically presentable members are simple or complex (in the senses described), or somewhere in-between.

A closely related problem is that of providing techniques for showing that a given structure does not have an automatic presentation. For instance, Delhommé [24] provides a necessary condition for a structure to have an automatic presentation, which implies, in particular, that the ordinal ω^ω does not (Corollary 3.52).

The following summary of the rest of this survey should give an indication of the main lines of research in the area.

1.1. Summary. Section 2 (Definitions) includes the definitions of regular relations and automatic presentations. The section ends with some common examples.

Section 3 (Properties) first covers general properties of automatically presentable structures, and later some specific topics.

²The model checking problem is, given a presentation of \mathcal{A} , a formula $\phi(\bar{x})$, and a tuple of parameters \bar{a} in \mathcal{A} , to decide whether or not $\mathcal{A} \models \phi(\bar{a})$.

Subsection 3.1 ‘Fundamental properties’ includes the decidability result (Theorem 3.2), closure under natural operations (Corollary 3.7), and the logical characterisation of automatically presentable structures (Theorem 3.8).

Subsection 3.2 ‘Extending first-order’ refers to extending the basic decidability result by certain additional quantifiers including ‘there exists infinitely many’ and ‘there exist k modulo m many’. This is done in the context of generalised quantifiers and order-invariance.

Subsection 3.3 ‘Automatic presentations’ offers a definition of when two presentations are equivalent, and gives a machine theoretic characterisation.

Subsection 3.4 ‘Intrinsic regularity’ presents the analogue of intrinsically computably enumerable relations (see [1]) for automatically presentable structures.

Subsection 3.5 ‘Restriction on growth’ presents some general ways of proving that a given structure has no automatic presentation. As illustration, the following structures are not automatically presentable: the free semigroup on $k > 1$ generators, Skolem arithmetic (\mathbb{N}, \times) , the random graph, and the ordinal ω^ω .

Subsection 3.6 ‘Isomorphism problem’ includes the proof that the isomorphism problem for automatically presented structures is Σ_1^1 -complete.

Section 4 (Classifications) is concerned with classifying the automatically presentable members of a given class of structures in relevant algebraic terms. There are many classes for which a complete classification is known for the restricted notion of automaticity requiring that only strings over a unary alphabet are used. In the general (non-unary) case, a complete classification is known for the classes of ordinals, Boolean algebras and fields. However, only partial classifications are known for the following classes: equivalence structures, linear orders, groups, and rings.

Section 5 (Open problems) contains a sample of problems whose solutions will likely require new ideas.

1.2. Scope and related work. Familiarity is assumed with the basics of finite automata, formal languages, and logic. Proofs will generally be sketched. The reader is referred to the literature for details: the original article [34]; the article [12] for an excellent reference of the fundamental results; and the theses [9], [51] and [3] which may serve as introductions to the area.

Automatically presentable structures can be generalised in several directions: for instance, by using finite automata on infinite strings [9, 12, 39], finite ranked trees [9, 5], finite unranked trees [40], or WMSO-interpretations (of dimension one) of trees with decidable WMSO-theory [21] (see Definition 3.4). The general theory goes through: decidability of the first-order theory and characterisations via interpretability in some structure. Problems such as classification and techniques for proving non-automaticity in these

more general settings are not dealt with in this survey. Consult [24, 21] for techniques showing that a structure has no finite-tree automatic presentation; and [39, 4] for the addition of generalised quantifiers ‘there exists uncountably many’, ‘there exists countably many’ and ‘there exists k modulo m many’ in ω -automatically presentable structures; and [21] for intrinsic regularity and proving non-automaticity in WMSO-interpreted structures.

Automatic groups [19] are not in the scope of this survey. See [14] for some remarks relating them to automatically presentable structures.

For the complexity of model-checking and related problems, consult [9, 12] for fragments of FO, and [43] for structures of bounded degree.

Definability issues (VC-dimension, quantifier elimination) in universal string- and tree-automatically presentable structures and their reducts can be found in [40, 5, 6].

1.3. Acknowledgement. I thank Valentin Goranko for discussions on intrinsic regularity, Lauri Hella for discussions on generalised quantifiers, André Nies for some corrections, Wolfgang Thomas for clarifying Trahtenbrot’s historical contribution, and Vince Bárány, Michael Benedikt and Alexander Raichev for their comments.

§2. Definitions.

2.1. Notation. Countable means finite or countably infinite. A *relational signature* τ is a countable sequence of symbols $(R_i)_i$ and corresponding arities r_i . A τ -structure $\mathcal{A} = (\mathcal{A}; (R_i^\mathcal{A})_i)$ consists of a countable set \mathcal{A} , called the *domain* of \mathcal{A} , written $\text{dom}(\mathcal{A})$, and for each i , a relation $R_i^\mathcal{A} \subseteq \mathcal{A}^{r_i}$, called an *atomic relation* of \mathcal{A} . When there is only one structure around, I may drop the superscript with the name of the structure.

Structures are written in script $\mathcal{A}, \mathcal{B}, \dots$ and their corresponding domains are written in capitals A, B, \dots . The substructure of relational \mathcal{A} on set $B \subseteq \mathcal{A}$ is written $\mathcal{A} \upharpoonright B$ or \mathcal{B} if there can be no confusion.

Signatures are assumed to be computable (the mappings $i \mapsto R_i$ and $i \mapsto r_i$ are computable), and to contain the equality symbol $=$, though this symbol may not be explicitly mentioned in the signature. For convenience, I sometimes write a structure containing functions, such as $(\mathbb{N}, +)$, but am implicitly referring to its *relational variant* obtained by replacing every function $f : B^k \rightarrow B$ by its graph $\{(\bar{x}, y) \in B^{k+1} \mid f(\bar{x}) = y\}$.

If unspecified, all formulas $\phi(\bar{x})$ (and associated notions like definability) are first-order and allow parameters. However, we will see extensions \mathcal{L} of first-order logic, particularly monadic second-order logic and extensions by generalised quantifiers.

An \mathcal{A} -formula is a formula over the signature of structure \mathcal{A} . The *relation in \mathcal{A} defined by $\Phi(\bar{x}, \bar{y})$ with parameters \bar{b}* , denoted by $\Phi^\mathcal{A}(\cdot, \bar{b})$, is defined as

$$\{(a_1, \dots, a_m) \mid \mathcal{A} \models \Phi(\bar{a}, \bar{b})\},$$

where \mathcal{A} is a τ -structure, $\Phi(\bar{x}, \bar{y})$ is an \mathcal{A} -formula with free variables $\bar{x} = (x_1, \dots, x_m)$, and \bar{b} a tuple from \mathcal{A} . To ease readability, I often relax the notation and write $\Phi^{\mathcal{A}}(\bar{b})$ or even $\Phi^{\mathcal{A}}$.

Familiarity is assumed with the basics from automata theory. To fix notation: symbols will usually be denoted a, b, \dots ; a finite alphabet of symbols by Σ ; finite strings by u, v, w, \dots ; the set of finite strings over Σ by Σ^* ; the set of infinite strings over Σ by Σ^ω ; the empty string by λ ; concatenation by \cdot as in $w \cdot v$, or simply by juxtaposition, as in wv ; the concatenation of a symbol a with itself n times by a^n (a^0 is defined as λ); the length of a string w by $|w|$; the strings in a set $A \subseteq \Sigma^*$ of length exactly n by $A^{=n}$, and those of length at most n by $A^{\leq n}$. These should not be confused with the set A^n of n -tuples from a set or domain A . A deterministic finite automaton M over alphabet Σ is of the form (Q, ι, Δ, F) where Q is a finite set of states, $\iota \in Q$ is the initial state, $\Delta: Q \times \Sigma \rightarrow Q$ is the transition function, and $F \subseteq Q$ is the set of accepting states.

The logarithmic and exponential functions are taken with base 2. In particular, the functions $\exp_k: \mathbb{N} \rightarrow \mathbb{N}$ are defined for $k \in \mathbb{N}$ recursively by $\exp_0(n) = n$ and $\exp_{k+1}(n) = 2^{\exp_k(n)}$. The subscript in \exp_1 will be dropped.

2.2. Synchronous finite automata. Consider a finite automaton as a restricted non-deterministic Turing machine: it has a read-only input-tape with one head that only moves in one direction. It has no work tape. By admitting more than one input-tape, say n many, each with its own head moving independently, the language computed by such a machine is an n -ary relation. The resulting relations are called *rational relations*. These multi-tape automata do not share the robustness of one-tape automata: they are not closed under the Boolean operations, and the nondeterministic rational relations strictly contain the deterministic rational relations.

This article deals with particular rational relations that do enjoy strong closure properties (Theorem 2.4), namely those recognisable by *synchronous n -tape automata* (also called letter-to-letter automata). The following informal description follows Eilenberg, Elgot, and Shepherdson [26]. A synchronous n -tape automaton can be thought of as a one-way Turing machine with n input-tapes. Each tape is regarded as semi-infinite having written on it a string over the alphabet Σ followed by an infinite succession of blanks, \perp symbols. The automaton starts in the initial state, reads simultaneously the first symbol of each tape, changes state, reads simultaneously the second symbol of each tape, changes state, etc., until it reads a blank on each tape. The automaton then stops and accepts the n -tuple of strings if it is in an accepting state. The set of all n -tuples accepted by the automaton is the relation recognised by the automaton. Since n -tape automata are simply 1-tape automata over a new alphabet, they can be determinised by the usual subset construction.

Instead of formalising this model of computation, we encode a tuple of strings from Σ^* as a single string over an expanded alphabet. For example, for $\Sigma = \{1\}$ the expanded alphabet is $\{\left(\begin{smallmatrix} 1 \\ 1 \end{smallmatrix}\right), \left(\begin{smallmatrix} 1 \\ \perp \end{smallmatrix}\right), \left(\begin{smallmatrix} \perp \\ 1 \end{smallmatrix}\right), \left(\begin{smallmatrix} \perp \\ \perp \end{smallmatrix}\right)\}$ and the string associated with the tuple $(1^m, 1^{m+1})$ is $\left(\begin{smallmatrix} 1 \\ 1 \end{smallmatrix}\right)^m \left(\begin{smallmatrix} 1 \\ \perp \end{smallmatrix}\right)$.

DEFINITION 2.1. Write Σ_\perp for $\Sigma \cup \{\perp\}$, where \perp is a symbol not in Σ . The *convolution of a tuple* $(w_1, \dots, w_n) \in (\Sigma^*)^n$ is the string $\otimes(w_1, \dots, w_n)$ over alphabet $(\Sigma_\perp)^n$, of length $\max_i |w_i|$, defined as follows. Its k th symbol is (a_1, \dots, a_n) where a_i is the k th symbol of w_i if $k \leq |w_i|$ and \perp otherwise. In particular, $\otimes(\lambda, \dots, \lambda) = \lambda$.

The *convolution of a relation* $R \subseteq (\Sigma^*)^n$ is the set $\otimes R = \{\otimes \overline{w} \mid \overline{w} \in R\}$; that is, $\otimes R \subseteq (\Sigma_\perp)^{n*}$ is the set of convolutions of all the tuples in R .

DEFINITION 2.2. A relation $R \subseteq (\Sigma^*)^n$ is called *synchronous rational*, or simply *regular*, if there is a finite automaton over alphabet $(\Sigma_\perp)^n$ recognising the convolution $\otimes R$.

In case $n = 1$, the convolution $\otimes R$ equals R , and so in this case the definition coincides with the traditional class of regular languages.

EXAMPLES 2.3. The following relations on Σ^* are regular.

- (i) The prefix relation \preceq_p .
- (ii) The equal length relation $\text{el}(w, v)$ if $|w| = |v|$.
- (iii) The longest common prefix relation, written functionally $x \sqcap y = z$.
- (iv) The prefix-lexicographic ordering (induced by a fixed ordering $<$ on Σ) defined by $x \leq_{\text{lex}} y$ if $x \preceq_p y$ or otherwise $za \preceq_p x$ and $zb \preceq_p y$ and $a < b$ where $z = x \sqcap y$.
- (v) The length-lexicographic ordering \leq_{llex} defined by $x \leq_{\text{llex}} y$ if $|x| < |y|$ or otherwise $|x| = |y|$ and $x \leq_{\text{lex}} y$.

The synchronous coding ensures that the regular relations have basic closure properties.

THEOREM 2.4. Let $R, S \subseteq \Sigma^{*n}$ be regular relations. Then the following relations are also regular:

- (i) union $R \cup S$, intersection $R \cap S$, relative complementation $R \setminus S$;
- (ii) projection $\{\overline{y} \mid (\exists x)R(x, \overline{y})\}$;
- (iii) instantiation $\{\overline{y} \mid R(w, \overline{y})\}$ for fixed $w \in \Sigma^*$;
- (iv) cylindrification $\{(x, \overline{y}) \mid x \in \Sigma^* \wedge R(\overline{y})\}$; and
- (v) permutation of the co-ordinates of R .

Moreover, there is an effective procedure that given automata for $\otimes R$ and $\otimes S$, constructs an automaton for the convolution of each of the resulting relations.

2.3. Automatic presentations. We are ready to define what it means for a structure to be described by a collection of automata. Although the definition is only for relational structures, recall the convention that we implicitly replace functions by their graphs.

DEFINITION 2.5. A (*finite-string*) *automatic presentation* of a relational structure $\mathcal{B} = (B; (R_i^{\mathcal{B}})_i)$ consists of a mapping μ and a tuple of automata $\overline{M} = (M_A, M_=, (M_i)_i)$ so that

- (i) M_A recognises a set $A \subseteq \Sigma^*$,
- (ii) the mapping $\mu: A \rightarrow \text{dom}(\mathcal{B})$ is surjective, and
- (iii) for every atomic relation $R_i^{\mathcal{B}}$ of \mathcal{B} (say of arity r_i), the relation

$$\mu^{-1}(R_i^{\mathcal{B}}) := \{(w_1, \dots, w_{r_i}) \in A^{r_i} \mid R_i^{\mathcal{B}}(\mu(w_1), \dots, \mu(w_{r_i}))\},$$

is regular; and its convolution is recognised by the automaton M_i .

Since by our convention there is a symbol for equality, the relation

$$\{(w_1, w_2) \in A^2 \mid \mu(w_1) =^{\mathcal{B}} \mu(w_2)\}$$

need also be regular; and its convolution is recognised by the automaton $M_=$.

Say that \mathcal{B} is *automatically presentable*. Depending on the focus, the *automatic presentation* may simply refer to the mapping μ or to the tuple of automata \overline{M} .

The induced quotient structure

$$(A; (\mu^{-1}(R_i^{\mathcal{B}}))_i) / \mu^{-1}(=^{\mathcal{B}})$$

is an *automatic copy* of \mathcal{B} .

If the signature of \mathcal{B} is infinite, we also require that the function mapping $i \in \mathbb{N}$ to (a code for) the automaton M_i be computable.

If $|\Sigma| = 1$, then we speak of a *unary-automatic presentation*.

The idea is that strings from A code elements of B (via the mapping μ) so that the induced relations $\mu^{-1}(R_i^{\mathcal{B}})$ are regular. In general, an element of B may have more than one code. However, if μ is also an injection, then every element of B is coded by a unique string of A . In this case, μ is called an *injective automatic presentation* of \mathcal{B} .

PROPOSITION 2.6. *Every automatically presentable structure \mathcal{B} has an injective automatic presentation v . Moreover, v can be chosen over a binary alphabet (that is, with $|\Sigma| = 2$).*

PROOF. Let $\mu: A \rightarrow B$ be an automatic presentation of \mathcal{B} over alphabet Γ . For the first item, it is sufficient to restrict A to a regular set A' so that $\mu: A' \rightarrow B$ is a bijection. To this end, define A' as the set of strings $v \in A$ such that v is the length-lexicographically least string in the set $\{w \in A \mid \mu(w) = \mu(v)\}$. It is straightforward to check that A' is regular.

For the second item, suppose $k = |\Gamma| > 2$ and let Σ be a binary alphabet. Fix some integer $l \geq \log_2 k$. The idea is to code each symbol $\gamma \in \Gamma$ by a distinct string $\alpha(\gamma) \in \Sigma^*$ of length l . Then extend α to Γ^* in the natural way (define $\alpha(uv) = \alpha(u)\alpha(v)$). The required presentation is $v: \alpha(A') \rightarrow B$ defined by $v(\alpha(v)) = \mu(v)$ for $v \in A'$. \dashv

2.4. Examples.

- (i) Every finite structure is automatically presentable.
- (ii) For $\Sigma = \{0, \dots, k-1\}$ ($k \in \mathbb{N}$), define the structure \mathcal{W}_k as

$$(\Sigma^*; (\sigma_a)_{a \in \Sigma}, \preceq_p, \text{el}).$$

Here $\sigma_a(w) = wa$, \preceq_p is the prefix relation, and $\text{el}(w, v)$ if $|w| = |v|$. It is automatically presentable.

- (iii) The structure

$$\mathcal{N}_k = (\mathbb{N}; +, |_k)$$

is automatically presentable where $+$ is the usual addition on the naturals \mathbb{N} , and $x|_k y$ means that $x = k^n$ for some $n \in \mathbb{N}$ and $y = mx$ for some $m \in \mathbb{N}$ (that is, x is a power of k and x divides y). One gets an automatic presentation for this structure by coding the natural numbers in base k (least significant digit first). Indeed, a finite automaton can check addition using the usual carry-bit procedure, and it is similarly straightforward to verify that the relation $|_k$ is regular.

- (iv) The configuration space of a Turing Machine N , considered as a directed graph, whose edges represent one step transitions of N , is automatically presentable. The idea is that an automaton can compute these transitions simply following a window of fixed size around the current position of the read-head.
- (v) The linear ordering of the rationals is automatically presentable. The structure with domain $\{0, 1\}^*$ and the binary relation $x \sqsubseteq_Q y$ if $(x \sqcap y)0$ is a prefix of x or $(x \sqcap y)1$ is a prefix of y (here $x \sqcap y$ denotes their longest common prefix) is an automatic copy.
- (vi) Every finitely generated Abelian group $(G; +)$ is automatically presentable. This is straightforward given the classification of these groups as finite sums of $(\mathbb{Z}, +)$ and finite cyclic groups.
- (vii) Every ordinal $(L; \leq)$ less than ω^ω is automatically presentable. This is straightforward given the classification of these ordinals as being of the form $\omega^{n_1} + \dots + \omega^{n_l}$ where $\omega > n_1 \geq n_2 \geq \dots \geq n_l$.
- (viii) The Boolean algebra $(\mathcal{B}_{\text{fin}})^n$ ($n \in \mathbb{N}$) is automatically presentable; here $\mathcal{B}_{\text{fin}} = (B_{\text{fin}}; \wedge, \vee, \neg)$ is the Boolean algebra of finite or co-finite subset of \mathbb{N} .

§3. Properties.

3.1. Fundamental properties. As a consequence of Theorem 2.4 one has the following result that may be called the fundamental theorem of automatically presentable structures.

THEOREM 3.1 (Definability). *For every automatic presentation μ , of structure \mathcal{A} say, every first order \mathcal{A} -formula (allowing parameters) defines a relation R with $\mu^{-1}(R)$ regular.*

Moreover there is an algorithm that from the automata of an automatic presentation, and a first-order formula allowing parameters, outputs an automaton for the convolution of the relation defined by the formula.

This is simply proved by structural induction on the formula defining the relation. Consequently:

THEOREM 3.2 (Decidability). *The first-order theory (allowing parameters) of each automatically presentable structure is decidable.*

Indeed, from any automatic presentation of \mathcal{A} , and sentence of the form $\exists x\Phi(x)$, one can check effectively whether or not the constructed automaton for $\Phi^{\mathcal{A}}(x)$ accepts any string at all.

Remark 3.3. We shall see in subsection 3.2 that the Definability and Decidability Theorems can be extended to include additional quantifiers such as ‘there exists infinitely many’ and ‘there exists k modulo m many’.

Another consequence of the Definability Theorem is that automatically presentable structures are closed under first-order interpretability.

DEFINITION 3.4. An interpretation of structure $\mathcal{B} = (B; (R_i^{\mathcal{B}})_i)$ in structure \mathcal{A} consists of the following \mathcal{A} -formulas,

- (i) a domain formula $\Delta(\bar{x})$,
- (ii) a relation formula $\Phi_{R_i}(\bar{x}_1, \dots, \bar{x}_{r_i})$ for each relation symbol R_i , and
- (iii) an equality formula $\epsilon(\bar{x}_1, \bar{x}_2)$,

where each $\Phi_{R_i}^{\mathcal{A}}$ is a relation on $\Delta^{\mathcal{A}}$, and $\epsilon^{\mathcal{A}}$ is a congruence on the structure $(\Delta^{\mathcal{A}}; (\Phi_{R_i}^{\mathcal{A}})_i)$, so that $(\Delta^{\mathcal{A}}; (\Phi_{R_i}^{\mathcal{A}})_i)/\epsilon^{\mathcal{A}}$ and \mathcal{B} are isomorphic.

Each of the tuples \bar{x}_i, \bar{x} contain the same number of variables, called the dimension of the interpretation.

We say that \mathcal{B} is interpretable in \mathcal{A} .

Also, say that \mathcal{B} is interpretable in \mathcal{A} with parameters \bar{a} if \mathcal{B} is interpretable in (\mathcal{A}, \bar{a}) .

In case \mathcal{B} has infinite signature, it is also required that the function sending i to Φ_{R_i} be computable.

To stress that all the formulas are first-order, we say that \mathcal{B} is *first-order interpretable* in \mathcal{A} . In this case, every first-order \mathcal{B} -formula ϕ can be translated into a first-order \mathcal{A} -formula ϕ' such that for all $\bar{a} \in \Delta^{\mathcal{A}}$,

$$\mathcal{B} \models \phi(\mu(\bar{a})) \iff \mathcal{A} \models \phi'(\bar{a}),$$

where μ is the isomorphism from the interpretation. The formula ϕ' is formed from ϕ by relativising the quantification to Δ , replacing every relation symbol R_i by its defining formula Φ_{R_i} , and replacing equality by ϵ .

We will have brief occasion to consider the case that the formulas are (weak) monadic second-order, whence we say that \mathcal{B} is (weak) monadic

second-order interpretable in \mathcal{A} .³ In this case, elements of \mathcal{B} are coded by (finite) subsets of \mathcal{A} . As an illustration, $(\mathbb{N}, +)$ is weak monadic second-order interpretable in (\mathbb{N}, S) via the map sending a finite set $X \subset \mathbb{N}$ to the natural number $\sum_{i \in X} 2^i$.

EXAMPLE 3.5. The structures \mathcal{N}_k and \mathcal{W}_l (see Example 2.4) are first-order interpretable in each other, for every $k, l \geq 2$.

The following proposition is immediate from the definitions and Theorem 3.1 [Definability].

PROPOSITION 3.6. [9] *If \mathcal{B} is first-order interpretable in \mathcal{A} (possibly with parameters), and \mathcal{A} is automatically presentable, then \mathcal{B} is also automatically presentable.*

The following corollary lists some useful closure properties of automatically presentable structures, some of which were already noted in [34]. It is worth mentioning that, more generally, Blumensath and Grädel [9, 12] show that automatically presentable structures are closed under Feferman-Vaught like products.

Recall that definable means first-order definable allowing parameters.

COROLLARY 3.7. *If structures \mathcal{A} and \mathcal{B} (with the same signature) are automatically presentable, then the following are also automatically presentable:*

- (i) *the expansion (\mathcal{A}, R) by any relation definable in \mathcal{A} ,*
- (ii) *the substructure of \mathcal{A} with any definable universe,*
- (iii) *the factorisation of \mathcal{A} by any definable congruence,*
- (iv) *the direct product $\mathcal{A} \times \mathcal{B}$, the disjoint union $\mathcal{A} \cup \mathcal{B}$,*
- (v) *the ω -fold disjoint union of \mathcal{A} , written $\cup_\omega \mathcal{A}$,*
- (vi) *the ordered sum $\mathcal{A} + \mathcal{B}$ and the ω -fold ordered sum $\Sigma_{i < \omega} \mathcal{A}$ (for this case \mathcal{A} and \mathcal{B} are ordered structures).*

Next is a logical characterisation of the automatically presentable structures.

THEOREM 3.8. *Let \mathcal{B} be a structure. Then the following are equivalent:*

- (i) *\mathcal{B} is automatically presentable.*
- (ii) *\mathcal{B} is first-order interpretable in \mathcal{W}_k for some, equivalently all, $k \geq 2$.*
- (iii) *\mathcal{B} is first-order interpretable in \mathcal{N}_k for some, equivalently all, $k \geq 2$.*
- (iv) *\mathcal{B} is weak monadic second-order interpretable in (\mathbb{N}, S) , where S is the usual successor function $n \mapsto n + 1$.*

Here is a brief history of this theorem. The equivalences (i)–(iii) are from Blumensath and Grädel [12] who introduced interpretability to characterise the automatically presentable structures. The main ideas arise from older

³These are called (*finite*) *set interpretations* in [21] if the interpretation also has dimension one. The present definition is not to be confused with the case that the formulas are (W)MSO but all free variables are first-order, sometimes also called (W)MSO-interpretations in the literature.

results stated for relations instead of structures: Büchi [16] and Elgot [27] prove that a set of tuples (A_1, \dots, A_n) of finite sets of natural numbers is weak monadic second-order definable in (\mathbb{N}, S) if and only if the corresponding n -ary relation of characteristic strings (a subset of $\{0, 1\}^{*n}$) is synchronous rational. The relationship between weak MSO and finite automata was also realised by Trahtenbrot [56]. Eilenberg, Elgot, and Shepherdson [26] prove that a relation $R \subseteq (\Sigma^*)^n$ is synchronous rational if and only if R is first-order definable in \mathcal{W}_k , where $k = |\Sigma| \geq 2$. The Büchi-Bruyère Theorem (proofs of which can be found in [44] and [57]) states that a relation $R \subseteq \mathbb{N}^n$ (coded in base $k \geq 2$ least significant digit first) is synchronous rational if and only if it is first-order definable in \mathcal{N}_k .

The proofs of these results, which are by now standard, usually go as follows. From formulas to automata one proceeds by structural induction on the given formula, using Theorem 2.4 for the logical operations. Conversely, starting with an automaton, one constructs a formula stating the existence of a successful run of the automaton (alternatively, [57] proceeds by induction on regular expressions).

We turn to unary-automatically presentable structures. An important example is the structure $\mathcal{U} = (\mathbb{N}; \leq, (\equiv_n)_{n \in \mathbb{N}})$ where $x \equiv_n y$ if x is congruent to y modulo n .

THEOREM 3.9. [9, 45] *A structure is automatically presentable over a unary alphabet if and only if it is first-order interpretable (via a 1-dimensional interpretation) in the structure $(\mathbb{N}; \leq, (\equiv_n)_{n \in \mathbb{N}})$.*

Blumensath [11] calls a structure \mathcal{A} *tree-interpretable* if there is a monadic second-order interpretation of dimension 1 of \mathcal{A} into \mathcal{T}_2 with the restriction that all the free variables in the interpreting formulas are first-order. Here $\mathcal{T}_2 = (\{0, 1\}^*; \sigma_0, \sigma_1)$ where $\sigma_\epsilon(w) = w\epsilon$ for all $w \in \{0, 1\}^*$ and $\epsilon \in \{0, 1\}$. Since \mathcal{T}_2 has decidable monadic second-order theory [49], so does \mathcal{A} .

Since \mathcal{U} is tree-interpretable, every unary-automatically presentable structure has decidable monadic second-order theory [10, Prop. 5]. Moreover, Blumensath proves that every tree-interpretable structure is automatically presentable (the converse is false because there are automatically presentable structures, such as the infinite grid, with undecidable monadic-second order theory). The tree-interpretable graphs coincide with other classes of infinite graphs that appear in the literature; see [10] for a discussion.

3.2. Extending first-order. The Definability Theorem can be strengthened to include order-invariantly definable formulas as well as certain additional quantifiers.

Order-invariance.

DEFINITION 3.10. Fix a signature τ and a new symbol \leq . A formula $\phi(\bar{x})$ in the signature $\tau \cup \{\leq\}$, is called ω -order invariant on a τ -structure \mathcal{A} , if for all tuples \bar{a} from A , and all linear orders \leq_1 and \leq_2 on A of order type ω (or $|A|$ if A is finite), we have that $(\mathcal{A}, \leq_1) \models \phi(\bar{a})$ if and only if $(\mathcal{A}, \leq_2) \models \phi(\bar{a})$.

The *relation in \mathcal{A} defined by the ω -order invariant ϕ* is the set of tuples \bar{a} from A such that $(\mathcal{A}, \leq) \models \phi(\bar{a})$ for some (and hence all) linear orders \leq on A of order type ω (or $|A|$ if A is finite).

Write $\mathcal{L}_{\omega-\text{inv}}(\mathcal{A})$ for the relations that are definable by ω -order invariant \mathcal{A} -formulas in logic \mathcal{L} .

Remark 3.11. Every automatic presentation μ of structure \mathcal{A} can be expanded to an automatic presentation of an expansion (\mathcal{A}, \leq) where \leq linearly orders A . For instance let $\mu^{-1}(\leq)$ be the regular linear order $<_{\text{lex}}$ restricted to the domain $\mu^{-1}(A)$. Observe that this linear order has order-type ω if A is infinite.

Thus we can replace FO by $\text{FO}_{\omega-\text{inv}}$ in Theorem 3.1 [Definability].

Generalised quantifiers. We briefly recall the definition of generalised quantifiers as introduced by Lindström [41].

DEFINITION 3.12. Fix a finite signature $\tau = (R_i)_{i \leq k}$, where R_i has associated arity r_i . A *quantifier* Q is a class of τ -structures closed under isomorphism. Let σ be another signature. Given σ -formulas $\Psi_i(\bar{x}_i, \bar{z})$ with $|\bar{x}_i| = r_i$ ($i \leq k$), the syntax $Q\bar{x}_1, \dots, \bar{x}_k(\Psi_1, \dots, \Psi_k)$ has the following meaning on a σ -structure \mathcal{A} :

$$(\mathcal{A}, \bar{a}) \models Q\bar{x}_1, \dots, \bar{x}_k(\Psi_1, \dots, \Psi_k) \text{ iff } (A; \Psi_1^{\mathcal{A}}(\cdot, \bar{a}), \dots, \Psi_k^{\mathcal{A}}(\cdot, \bar{a})) \in Q,$$

where $\Psi^{\mathcal{A}}(\cdot, \bar{a})$ is the relation defined in \mathcal{A} by Ψ with parameters \bar{a} . The *arity* of a quantifier is the maximum of the r_i s. A quantifier is n -ary if its arity is at most n .

The *relation defined (in \mathcal{A}) by this formula* is the set of tuples \bar{a} from A such that $(\mathcal{A}, \bar{a}) \models Q\bar{x}_1, \dots, \bar{x}_k(\Psi_1, \dots, \Psi_k)$.

A collection of quantifiers is denoted by \mathbf{Q} , and the extension of first-order logic by the quantifiers in \mathbf{Q} is written $\text{FO}[\mathbf{Q}]$.

As an illustration, ‘there exists’ is given by the unary quantifier

$$\{(A; X) \mid \emptyset \neq X \subseteq A\}.$$

Here are some quantifiers that will feature in this section:

EXAMPLES 3.13. (i) The unary quantifier ‘there exists infinitely many’, written \exists^∞ , is the class of structures $(A; X)$ where X is an infinite subset of A .

(ii) The unary *modulo quantifier* ‘there are k modulo m many’ (here $0 \leq k < m$), written $\exists^{(k,m)}$, is the class of structures $(A; X)$ where X contains k modulo m many elements. Write \exists^{mod} for the collection of modulo quantifiers.

(iii) The unary *Härtig quantifier* is the class of structures $(A; P, Q)$ where $P, Q \subseteq A$ and $|P| = |Q|$.

- (iv) The *cardinality quantifiers*: Every set $C \subseteq (\mathbb{N} \cup \{\infty\})^n$ induces the unary quantifier $Q_C = \{(A; P_1, \dots, P_n) \mid (|P_1|, \dots, |P_n|) \in C\}$. In fact, a given unary quantifier over signature $(R_i)_{i \leq k}$ is identical to some cardinality quantifier with $n = 2^k$.
- (v) The binary *reachability quantifier* is the class of structures of the form $(A; E, \{c_s\}, \{c_f\})$ where $E \subseteq A^2$, $c_s, c_f \in A$, and there is a path in the directed graph $(A; E)$ from c_s to c_f .
- (vi) The k -ary *Ramsey quantifier* $\exists^{k\text{-ram}}$ is the class of structures $(A; E)$, $E \subseteq A^k$, for which there is an infinite $X \subseteq A$ such that for all pairwise distinct $x_1, \dots, x_k \in X$, $E(x_1, \dots, x_k)$.

Of course the generalised quantifiers that interest us most are the ones, like \forall and \exists , that preserve regularity; meaning, loosely, that quantifying over regular relations yields regular relations.

DEFINITION 3.14. Let Q be a quantifier with signature $\tau = (R_i)_{i \leq k}$, where R_i has associated arity r_i . Say that the quantifier *preserves regularity* if for every $n \in \mathbb{N}$, and every automatic presentation μ of a structure \mathcal{A} , every formula

$$Q\bar{x}_1, \dots, \bar{x}_k(\Psi_1^{\mathcal{A}}(\bar{x}_1, \bar{z}), \dots, \Psi_k^{\mathcal{A}}(\bar{x}_k, \bar{z}))$$

defines a relation R in \mathcal{A} so that $\mu^{-1}(R)$ is regular (here $\bar{z} = (z_1, \dots, z_n)$ and the Ψ_i s are first-order \mathcal{A} -formulas).

Moreover, say that Q *preserves regularity effectively* if an automaton for $\mu^{-1}(R)$ can effectively be constructed from the automata of the presentation and the formulas Ψ_i .

Remark 3.15. Since every automatic presentation restricts (effectively) to an injective automatic presentation (proof of Proposition 2.6), to show that Q preserves regularity (effectively) it is sufficient to satisfy the above definition with ‘injective automatic presentation’ replacing ‘automatic presentation’.

EXAMPLE 3.16. [9] The quantifier \exists^∞ preserves regularity effectively. Let μ be an injective automatic presentation of structure \mathcal{A} , and $\Psi(x, \bar{z})$ a first-order \mathcal{A} -formula. Then the relation defined in \mathcal{A} by $(\exists^\infty x)\Psi(x, \bar{z})$ is equal to the relation defined by the ω -order invariant formula

$$(\forall w)(\exists x)[w < x \wedge \Psi(x, \bar{z})].$$

Here is a non-example.

EXAMPLE 3.17. The reachability quantifier is not regularity preserving. For otherwise, by Example 2.4 (4), the set of starting configurations that drive a given Turing Machine to a halting state would be regular, and hence computable.

Remark 3.18. In our new terminology, Theorem 3.1 [Definability] says that \exists and \forall preserve regularity effectively. Moreover if every quantifier in a collection \mathbf{Q} preserves regularity effectively, then we can replace FO by $\text{FO}[\mathbf{Q}]$ in Theorem 3.1 [Definability] and Theorem 3.2 [Decidability]. If quantifiers in \mathbf{Q} simply preserve regularity, then we can replace FO by $\text{FO}[\mathbf{Q}]$ in Proposition 3.6, Corollary 3.7 and parts two and three of Theorem 3.8.

THEOREM 3.19. [36] *Every quantifier in \exists^{mod} preserves regularity effectively.*

PROOF. Let μ be an injective automatic presentation of structure \mathcal{D} , and let $\Phi(y_1, \dots, y_n)$ denote

$$\exists^{(k,m)} x \Psi(x, y_1, \dots, y_n),$$

for fixed $k, m \in \mathbb{N}$ and first-order \mathcal{D} -formula Ψ . Let \mathcal{A} , with $A \subseteq \Sigma^*$ be the automatic copy of \mathcal{D} induced by the presentation. The aim is to show that the relation defined in \mathcal{A} by Φ is regular, and to construct the automaton effectively from the automata of the presentation.

First, the required automaton should not accept those \bar{y} for which there are infinitely many x with $\mathcal{A} \models \Psi(x, \bar{y})$. So, define $\Psi_f(x, \bar{y})$ to be

$$\Psi(x, \bar{y}) \wedge \neg \exists^\infty x \Psi(x, \bar{y}).$$

Now, it is sufficient to construct an automaton, which will be called \mathbf{B}' , that recognises the relation defined in \mathcal{A} by $\exists^{(k,m)} x \Psi_f(x, \bar{y})$. The following property of Ψ_f will be used: For every tuple \bar{y} , there are finitely many x with $\mathcal{A} \models \Psi_f(x, \bar{y})$.

Let $\mathbf{B} = (Q, \iota, \delta, F)$ be a deterministic automaton recognising the convolution of $(\Psi_f)^\mathcal{A}$. The idea is that on input \bar{y} , the required automaton, \mathbf{B}' , will count, modulo m , the number of accepting paths of \mathbf{B} on inputs of the form (x, \bar{y}) for $x \in \Sigma^*$. Here is some notation to help describe the construction.

Notation: For states $q, q' \in Q$, and input $\bar{y} \in \Sigma^{*n}$ define $\#(q, \otimes \bar{y}, q')$ to be the number of strings $x \in \Sigma^*$ with $|x| = |\otimes \bar{y}|$, such that there is a path in \mathbf{B} labeled $\otimes(x, \bar{y})$ from state q to state q' . For $S \subseteq Q$ define $\#(S, \otimes \bar{y}, q')$ to be the sum $\sum_{q \in S} \#(q, \otimes \bar{y}, q')$.

More to the point then, the states of \mathbf{B}' are of the form (S_1, \dots, S_m) where $S_i \subseteq Q$. If \mathbf{B}' is in state (S_1, \dots, S_m) after reading input \bar{y} , then S_i will consist of those states q of \mathbf{B} for which $\#(\iota, \otimes \bar{y}, q)$ is congruent to i modulo m . Finally, once $\otimes \bar{y}$ has been completely read, the automaton needs to account for more input to \mathbf{B} of the form $\otimes(x', \lambda, \dots, \lambda)$.

Definition: The required automaton $\mathbf{B}' = (Q', \iota', \Delta', F')$ over alphabet $(\Sigma_\perp)^n$ is defined as follows.

- (i) The state set Q' is $\prod_{1 \leq i \leq m} \mathbb{P}(Q)$, where $\mathbb{P}(Q)$ denotes the powerset of Q .
- (ii) The initial state ι' is $\{\iota\} \times \prod_{2 \leq i \leq m} \{\emptyset\}$.

- (iii) For a state $T = (T_1, \dots, T_m) \in Q'$ and input symbol $\sigma \in (\Sigma_\perp)^n$ define the transition function $\Delta'(T, \sigma) = (S_1, \dots, S_m)$ as follows. For every set S_i and $q \in Q$, let $q \in S_i$ if and only if

$$\sum \{j \times \#(T_j, \sigma, q) \mid 1 \leq j \leq m\} \equiv i \pmod{m}.$$

- (iv) Let $(S_1, \dots, S_m) \in F'$ if and only if

$$\sum_{f \in F} \sum_{r < |Q|} \sum_{1 \leq j \leq m} j \times \#(S_j, (\{\perp\}^n)^r, f) \equiv k \pmod{m}.$$

Here $(\{\perp\}^n)^r$ is the concatenation of r copies of the new symbol $(\perp, \dots, \perp) \in (\Sigma_\perp)^n$.

Note that the automaton is deterministic.

Correctness: Let $w = \otimes \bar{y}$ be the input to \mathcal{B}' and let (S_1, \dots, S_m) be the state $\Delta'(l', w)$. The following can be proved by induction on $|w|$:

- (i) If $|w| \geq 1$, then for each S_i ,

$$q \in S_i \text{ if and only if } \#(l, w, q) \equiv i \pmod{m}.$$

- (ii) $(S_1, \dots, S_m) \in F'$ if and only if

$$\sum_{f \in F} \sum_{r \in \mathbb{N}} \#(l, w \cdot (\{\perp\}^n)^r, f) \equiv k \pmod{m}.$$

Finally, note that the index $r \in \mathbb{N}$ can be restricted to $0 \leq r < |Q|$, since the number of strings x with $\Psi_f(x, \bar{y})$ is finite. \dashv

A result in [37] says that the set of extendible nodes of an automatically presentable tree $(T; \preceq)$ is regular. We adapt the proof for the next Theorem.

THEOREM 3.20. *Each k -ary Ramsey quantifier preserves regularity effectively.*

PROOF. Note that the 1-Ramsey quantifier is identical to ‘there exists infinitely many’. We illustrate the proof for $k = 2$, the general case being similar.

Let μ be an injective automatic presentation of some structure \mathcal{D} , and $\Psi(x, y, z_1, \dots, z_n)$ a first-order \mathcal{D} -formula. Let \mathcal{A} be an automatic copy of \mathcal{D} with $A \subseteq \Sigma^*$. The aim is to show constructively that the binary relation defined in \mathcal{A} by the formula

$$\exists^{2\text{-ram}} xy \Psi(x, y, \bar{z}),$$

is regular.

Now, $\mathcal{A} \models \exists^{2\text{-ram}} xy \Psi(x, y, \bar{z})$ if and only if there exists an infinite string $\alpha \in \Sigma^\omega$ and an infinite set $I \subseteq \mathbb{N}$:

- (i) for every $i \in I$, there is a string $w_i := p_i t_i$ where p_i is the initial prefix of α of length i , and t_i is a non-empty string;
- (ii) and for every $i \neq j \in I$, it holds that $w_i \neq w_j$ and $\Psi(w_i, w_j, \bar{z})$.

Since this expression quantifies over infinite strings, we will use automata operating over infinite strings, so called ω -automata or Büchi-automata. Here is a brief reminder of what this means. A Büchi automaton has the same form (Q, ι, Δ, F) as a non-deterministic finite automaton. A run on an infinite string is accepting if some accepting state occurs infinitely in the run. A relation $R \subseteq (\Sigma^\omega)^m$ is *Büchi recognisable* if there is a Büchi automaton recognising the convolution of R , namely the relation $\otimes R \subseteq (\Sigma^m)^\omega$ where the i th symbol of $\otimes R$ is defined as $(\sigma_1, \dots, \sigma_m)$ where σ_j is the i th symbol of the j th component of R . The languages recognised by Büchi automata satisfy an analogous version of Theorem 2.4. For details, see for instance [55].

We need to mark the boundaries as given by I . Introduce a new symbol \wr for this purpose, and let $\Delta = \Sigma \cup \{\wr\}$. Call a pair (s, t) of infinite strings $s, t \in \Delta^\omega$ *good* if s can be expressed as $s_0 \wr s_1 \wr s_2 \wr \dots$ and t as $t_0 \wr t_1 \wr t_2 \wr \dots$, where $s_i, t_i \in \Sigma^*$ and $|s_i| = |t_i| > 0$ for each i . In this case, say that every string of the form $s_0 s_1 \dots s_l t_{l+1}$, for $l \in \mathbb{N}$, is *on* the good pair (s, t) .

Now define the language $R \subseteq (\Delta^\omega)^{2+n}$ as consisting of tuples of the form (s, t, u_1, \dots, u_n) with the following properties:

- (i) (s, t) is a good pair;
- (ii) every u_i is of the form $z_i \{\wr\}^\omega$ for some $z_i \in \Sigma^*$;
- (iii) and if x and y are distinct strings on (s, t) , then $\Psi(x, y, z_1, \dots, z_n)$ holds.

Then $\mathcal{A} \models \exists^{2\text{-ram}} xy \Psi(x, y, \bar{z})$ if and only if there exists (s, t) so that $R(s, t, z_1 \{\wr\}^\omega, \dots, z_n \{\wr\}^\omega)$.

To finish the proof, one can show that R is Büchi recognisable. Moreover, since Büchi automata are closed under projection, there is a Büchi automaton that accepts $(z_1 \{\wr\}^\omega, \dots, z_n \{\wr\}^\omega)$ if and only if \mathcal{A} models $\exists^{2\text{-ram}} xy \Psi(x, y, \bar{z})$. This automaton can be transformed into a finite automaton recognising the relation defined in \mathcal{A} by $\exists^{2\text{-ram}} xy \Psi(x, y, \bar{z})$. \dashv

As an aside, we now show that one can extract from this proof an automatic version of Ramsey's Theorem.

Recall that the infinitary version of Ramsey's Theorem (for k -tuples and two colours) says that if D is infinite and $E \subseteq D^k$, then there exists an infinite monochromatic set $X \subseteq D$; namely, either $E(\bar{x})$ for all pairwise distinct $x_1, \dots, x_k \in X$, or $\neg E(\bar{x})$ for all pairwise distinct $x_1, \dots, x_k \in X$.

PROPOSITION 3.21. *Let μ be an automatic presentation of a structure $\mathcal{D} = (D; E)$, with D infinite and $E \subseteq D^k$. Then there exists an infinite monochromatic set $X \subseteq D$ with $\mu^{-1}(X)$ regular.*

PROOF. For simplicity we consider the case $k = 2$. By Ramsey's Theorem there is a monochromatic set $X \subseteq D$. Suppose that $E(x, y)$ for all $x \neq y \in X$ (the other case is symmetric).

Let $\mathcal{A} = (A; E)$ be the automatic copy of \mathcal{D} induced by μ . By the previous proof, the set of good pairs (s, t) such that $x \neq y \in A$ on (s, t) implies

$E(x, y)$, is a Büchi recognisable relation, say R . If $R(s, t)$ then the set of strings $w \in A$ that are on (s, t) forms an infinite monochromatic set.

By assumption $R(\cdot, \cdot)$ is non-empty. Since every Büchi automaton accepting some string also accepts some ultimately periodic string, we get that R contains some ultimately periodic $\otimes(s', t') \in (\Delta^2)^\omega$. So both s' and t' are ultimately periodic. This pair (s', t') can be transformed into a finite automaton accepting all the strings $w \in A$ that are on (s', t') . \dashv

DEFINITION 3.22. Write $\mathcal{Q}_n^{\text{reg}}$ for the collection of n -ary generalised quantifiers that preserve regularity.

For instance, $\mathcal{Q}_1^{\text{reg}}$ contains the usual quantifiers \exists, \forall , as well as the modulo quantifiers \exists^{mod} and \exists^∞ ; and $\mathcal{Q}_n^{\text{reg}}$ contains $\exists^{\text{n-ram}}$.

Problem 3.23. Classify the quantifiers in $\mathcal{Q}_n^{\text{reg}}$, for each n .

The following general definition will allow us to compare the expressive strength of quantifiers.

DEFINITION 3.24. Let Q be a quantifier, \mathcal{Q} a collection of quantifiers, and τ the signature of Q . Say that Q is *definable in \mathcal{Q}* if there is a sentence θ over the signature τ in the logic $\text{FO}[\mathcal{Q}]$ with $Q = \{\mathcal{A} \mid \mathcal{A} \models \theta\}$.

For instance, a structure $(A; X)$ satisfies the sentence $\neg[\exists^{(0,2)}zX(z) \vee \exists^{(1,2)}zX(z)]$ if and only if X is infinite. So, \exists^∞ is definable in $\{\exists^{(0,2)}, \exists^{(1,2)}\}$.

As a first step, we characterise the unary quantifiers that preserve regularity.

PROPOSITION 3.25. Every quantifier in $\mathcal{Q}_1^{\text{reg}}$ is definable in \exists^{mod} .

PROOF. Every unary quantifier is equal to some cardinality quantifier where $C \subseteq (\mathbb{N} \cup \{\infty\})^r$ (Example 3.13 (4)). So suppose that \mathcal{Q}_C preserves regularity. Since \exists^∞ is definable in \exists^{mod} , we may assume that $C \subseteq \mathbb{N}^r$. Let $\Sigma = \{1, \dots, r\}$. Write $\#_i(z)$ for the number of occurrences of the symbol i in the string $z \in \Sigma^*$. Then the set

$$C' = \{z \in \Sigma^* \mid (\#_1(z), \dots, \#_r(z)) \in C\}$$

is regular since it is definable as $\mathcal{Q}_C \overline{x}(M_1(x_1, z), \dots, M_r(x_r, z))$, where for each $i \leq r$, M_i is the regular binary relation of those pairs (x, z) such that $x \in 11^*$ and z has the symbol i in position $|x|$.

Now if C' is regular, then C is recognisable - meaning that C is the union of classes of a congruence on $(\mathbb{N}^r, +)$ of finite index.

By Mezei's description of the recognisable subsets of a product of monoids (see for instance [8]), C is recognisable if and only if it is a finite union of sets of the form $X_1 \times \dots \times X_r$, where each $X_i \subseteq \mathbb{N}$ is ultimately periodic. In other words, \mathcal{Q}_C is definable in \exists^{mod} . \dashv

EXAMPLE 3.26. The Härtig quantifier does not preserve regularity. Indeed, the previous proof says that this is because one could use it to define, in a

suitable automatically presentable structure, the (non-regular) set of strings $x \in \{0, 1\}^*$ for which x has the same number of 0s as 1s.

What about $\mathcal{Q}_n^{\text{reg}}$ for $n \geq 2$? I do not know of a characterisation as for $n = 1$. Here is a simple example separating $\mathcal{Q}_2^{\text{reg}}$ from $\mathcal{Q}_1^{\text{reg}}$.

EXAMPLE 3.27. Define the quantifier \exists^ρ stating that ‘ $(A; \psi(\cdot, \cdot, \bar{z}))$ is an equivalence structure with infinitely many infinite classes’.

Then \exists^ρ is in $\mathcal{Q}_2^{\text{reg}}$, since it is $\text{FO}_{\omega-\text{inv}}(\exists^\infty)$ definable.

However, it is not expressible in first-order plus all unary quantifiers over the class of automatically presentable structures. This can be seen by checking that for every $r \in \mathbb{N}$, the Duplicator has a winning strategy in the 1-bijective r -round game⁴ between the following structures: \mathcal{A}_r comprises exactly r infinite classes, and \mathcal{B}_r comprises infinitely many infinite classes.

3.3. Automatic presentations. An automatically presentable structure \mathcal{B} has many automatic presentations. Here we look at the relationship amongst these presentations, and in the next subsection at their relationship to definability in the structure.

For a given automatic presentation $\mu: A \rightarrow B$ of \mathcal{B} , write $\mu(\text{Reg})$ for the collection of relations

$$\{\mu(R) \mid R \subseteq A^r \text{ is a regular relation, } r \in \mathbb{N}\}.$$

DEFINITION 3.28. [2] Let \mathcal{B} be an automatically presentable structure. Call two automatic presentations μ, ν of \mathcal{B} *equivalent* if $\mu(\text{Reg}) = \nu(\text{Reg})$.

EXAMPLE 3.29. Fix an automatic presentation μ of an infinite structure \mathcal{A} . There are infinitely many equivalent presentations ν so that the binary relation $\{(u, v) \mid \mu(u) = \nu(v)\}$ that translates between the presentations is not regular. For instance, for $c \notin \Sigma$ and $n \in \mathbb{N}$, let ν_n be the presentation sending $a_0 c^n a_1 c^n \dots a_k c^n$ to $\mu(a_0 a_1 \dots a_k)$ for $a_0 \dots a_k \in \text{dom}(\mu)$. Here c^n denotes the concatenation of n many c symbols.

Proposition 2.6 says that we can transform an arbitrary automatic presentation into one that is injective and over a binary alphabet. Its proof yields an equivalent presentation.

PROPOSITION 3.30. *Let μ be an automatic presentation of \mathcal{B} . Then there is an equivalent presentation ν of \mathcal{B} with the following properties:*

- (i) ν is injective, and
- (ii) ν is a presentation over a binary alphabet $|\Sigma| = 2$.

⁴The n -bijective games, introduced by Hella [29], are Ehrenfeucht–Fraïssé like games characterising definability in first-order logic extended by all n -ary generalised quantifiers. Here is a brief description: in round i the Duplicator chooses a bijection $b_i: A \rightarrow B$, and the spoiler answers with a set $C_i \subseteq A$ where $|C_i| \leq n$. Duplicator wins the r -round game if $\cup_{j \leq r} b_j \upharpoonright C_j$ is a partial isomorphism from \mathcal{A} to \mathcal{B} .

Bárány has characterised equivalence of presentations in automata theoretic terms. For the sake of completeness, here is the definition of semi-synchronous rational relation, which can be thought of as a relation recognised by a multi-tape automaton where each read-head advances at a different, but still constant, speed.

DEFINITION 3.31. Fix a finite alphabet Σ and a vector of positive integers $\underline{m} = (m_1, \dots, m_r)$. Let \perp be a symbol not in Σ . For each component m_i introduce the alphabet $(\Sigma_\perp)^{m_i}$. The *\underline{m} -convolution of a tuple* $(w_1, \dots, w_r) \in (\Sigma^*)^r$ is formed as follows: First, consider the intermediate string $(w_1 \perp^{a_1}, \dots, w_r \perp^{a_r})$ where the a_i are minimal such that there is some $k \in \mathbb{N}$ so that for all i , $|w_i| + a_i = km_i$. Second, partition each component $w_i \perp^{a_i}$ into k -many blocks of size m_i , and view each block as an element of $(\Sigma_\perp)^{m_i}$. Thus the string $\otimes_{\underline{m}}(w_1, \dots, w_r)$ is formed over alphabet $(\Sigma_\perp)^{m_1} \times \dots \times (\Sigma_\perp)^{m_r}$.

The *\underline{m} -convolution of a relation* $R \subseteq (\Sigma^*)^r$ is the set $\otimes_{\underline{m}}R$ defined as $\{\otimes_{\underline{m}}\overline{w} \mid \overline{w} \in R\}$.

Call R *\underline{m} -synchronous rational* if there is a finite automaton recognising $\otimes_{\underline{m}}R$.

Call R *semi-synchronous* if it is \underline{m} -synchronous rational for some \underline{m} .

In particular, a $(1, \dots, 1)$ -synchronous rational relation R is synchronous rational (Definition 2.2).

EXAMPLE 3.32. In the proof of Proposition 2.6, the coding $\alpha \subseteq \Gamma^* \times \Sigma^*$ is semi-synchronous. Similarly, the translation in Example 3.29 is semi-synchronous.

THEOREM 3.33. [2] *Two automatic presentations μ and ν of a structure \mathcal{B} are equivalent if and only if the relation $\{(u, v) \mid \mu(u) = \nu(v)\}$ is semi-synchronous.*

Here are a few words about the proof. The ‘if’ part follows from the basic property that the composition of a semi-synchronous relation with a regular relation is regular. On the other hand, for the ‘only if’ part it is sufficient to consider injective presentations, and prove that the translation $\mu^{-1}\nu: \text{dom}(\nu) \rightarrow \text{dom}(\mu)$, which by assumption preserves the regularity and non-regularity of relations on $\text{dom}(\nu)$, is semi-synchronous. This is achieved by a series of transformations, and is nicely presented in [3].

EXAMPLES 3.34. (i) With a little more work, one can show that each of the structures \mathcal{W}_k and \mathcal{N}_k (for $k \geq 2$) has exactly one presentation up to equivalence [2, 3].

(ii) Presburger arithmetic $(\mathbb{N}; +)$ has infinitely many non-equivalent presentations. This follows from a result of Büchi [16] saying that the set of powers of $n \geq 2$, written in base $m \geq 2$ coding, is regular if and only if $n^p = m^q$ for some positive integers p and q .

3.4. Intrinsic regularity. This subsection looks at the relationship between regularity and definability in automatically presentable structures. The following definition parallels that of the intrinsically computably enumerable relations; see Ash and Nerode [1]. Loosely, a relation $R \subseteq B^n$ (not assumed to be an atomic relation of \mathcal{B}) is called *intrinsically regular in \mathcal{B}* if it is regular in every automatic copy of \mathcal{B} .

DEFINITION 3.35. Fix an automatically presentable structure \mathcal{B} . Call a relation $R \subseteq B^n$ *intrinsically regular in \mathcal{B}* if for every automatic presentation μ of \mathcal{B} , the relation $\mu^{-1}(R)$ is regular (or in the terminology of the previous section, that $R \in \mu(\text{Reg})$).

Denote by $\text{IR}(\mathcal{B})$ the set of intrinsically regular relations in \mathcal{B} .

Problem 3.36. Can we capture intrinsic regularity using definability? For instance, by Remarks 3.11 and 3.18, for every automatically presentable structure \mathcal{B} we see that $\text{FO}[\cup_n Q_n^{\text{reg}}]_{\omega-\text{inv}}(\mathcal{B}) \subseteq \text{IR}(\mathcal{B})$. Is there equality here?

IR in some specific structures. We can describe $\text{IR}(\mathcal{B})$ for some specific automatically presentable \mathcal{B} s.

PROPOSITION 3.37.

- (i) $\text{IR}(\mathbb{N}, +, |_m) = \text{FO}(\mathbb{N}, +, |_m)$, for $m > 1$.
- (ii) $\text{IR}(\mathbb{N}, +) = \text{FO}(\mathbb{N}, +)$.
- (iii) $\text{IR}(\mathbb{N}, \leq) = \text{FO}[\exists^{\text{mod}}](\mathbb{N}, \leq)$.

The first item follows from the fact that (the base m coding of) a relation is regular only if it is first-order definable in $(\mathbb{N}, +, |_m)$ (see Theorem 3.8).

The second item follows from the Cobham-Semenov Theorem: R is first-order definable in $(\mathbb{N}, +)$ if for some $m, n \in \mathbb{N}$, R in base m is regular (as a subset of $\{0, 1, \dots, m-1\}^*$) and R in base n is regular, and $m^p \neq n^q$ for all positive integers p, q (see [15] for a discussion).

The third item follows from the characterisation of the structures with unary-automatic presentations (Theorem 3.9).

The proof of the next theorem, due to Frank Stephan, is technical and can be found in [51].

THEOREM 3.38. *For every $k \geq 2$, there is an automatic copy of (\mathbb{N}, S) in which the image of the set $\{n \in \mathbb{N} \mid k \text{ divides } n\}$ is not regular.*

Here S is the usual successor function $n \mapsto n+1$. A little work establishes the following corollary.

COROLLARY 3.39. *A unary relation $R \subseteq \mathbb{N}$ is intrinsically regular for the structure (\mathbb{N}, S) if and only if R is in $\text{FO}(\mathbb{N}, S)$.*

The proof of Theorem 3.38 may be adapted to yield automatic presentations with pathological properties.

A *cut* of the structure (\mathbb{Z}, S) is a set of the form $\{x \in \mathbb{Z} \mid x \geq n\}$ where $n \in \mathbb{Z}$ is fixed.

COROLLARY 3.40. *There is an automatic copy of (\mathbb{Z}, S) in which no cut is regular.*

COROLLARY 3.41. *There is an automatic copy of a graph with exactly two connected components, each isomorphic to (\mathbb{N}, S) , and neither regular.*

3.5. Restriction on growth. How can one prove that a given structure is not automatically presentable? This section provides some properties of automatically presentable structures, that in the contrapositive can be used to establish that a given structure has no automatic presentation. I first mention two methods based on results that we have already seen.

First, Theorem 3.2 says that the first-order theory of an automatically structure is decidable. This settles, for instance, that arithmetic $(\mathbb{N}, +, \times)$ is not automatically presentable. A closer approach would be to note that the proof of the theorem gives an upper bound of $\exp_{k-1}(c^n)$ for the space complexity of deciding the Σ_k (or Π_k) fragment of the first-order theory of an automatically presentable structure (here c is the size of the automata in the presentation, and n the size of the input sentence).

Second, if it has already been established that \mathcal{A} is not automatically presentable, then no structure that interprets \mathcal{A} is automatically presentable (Proposition 3.6).

We now turn to some finer techniques. These share the idea that in an automatically presentable structures, certain functions that count (elements, definable sets, etc.) cannot grow too fast.

A *locally-finite* relation R parameterised by $k, l \in N$, is one such that $R \subseteq A^{k+l}$ and for every tuple \bar{x} of size k there are at most a finite number of tuples \bar{y} of size l such that $(\bar{x}, \bar{y}) \in R$. The canonical example of a locally-finite relation is the graph of a function.

The following proposition says, in a simple case, that if (the graph of) a function f is regular, then for every x , the value $|f(x)| - |x|$ is bounded above by the number of states of an automaton for f . Although the proposition has an easy proof, it is an important tool.

PROPOSITION 3.42. *Suppose that locally-finite $R \subseteq A^{k+l}$ is regular. There exists a constant p , that depends only on the automaton for R , such*

$$\max\{|y| \mid y \in \bar{y}\} - \max\{|x| \mid x \in \bar{x}\} \leq p$$

for every $(\bar{x}, \bar{y}) \in R$.

PROOF. Take the simple case that R is the graph of a (partial) function $f : A \rightarrow A$. Suppose an automaton recognising $\otimes R$ has p states. Assume, aiming for a contradiction, that $|f(x)| > |x| + p$ for some x in the domain of f . Then the run of $\otimes(x, f(x))$, after reading the first $|x|$ symbols, must repeat a state. This implies that the automaton also accepts infinitely many strings of the form $\otimes(x, \cdot)$. \dashv

We will now see a useful way of iterating Proposition 3.42.

DEFINITION 3.43. Start with an automatic copy \mathcal{A} of a structure that includes functions f_1, \dots, f_k of arities r_1, \dots, r_k respectively. Let $D \subseteq A$ be regular, listed as d_i so that $|d_i| \leq |d_{i+1}|$.

Define the n th growth level, written $G_n(D)$, inductively by $G_1(D) = \{d_1\}$ and

$$\begin{aligned} G_{n+1}(D) = & \{d_{n+1}\} \cup G_n(D) \cup \\ & \cup_{i \leq k} \{f_i(x_1, \dots, x_{r_i}) \mid x_j \in G_n(D) \text{ for } 1 \leq j \leq r_i\}. \end{aligned}$$

We are interested in how fast $|G_n(D)|$ grows as a function of n . For example, consider the free semigroup (Σ^*, \cdot) with generating set $\Sigma = \{d_1, \dots, d_m\}$. For $m \geq 2$, since $G_m(\Sigma) \supseteq \Sigma$, the set $G_{m+n}(\Sigma)$ includes all strings over Σ of length at most 2^n ; thus the cardinality of $G_{m+n}(\Sigma)$ is at least a double exponential, namely $\exp_2(n)$.

PROPOSITION 3.44 ([12], cf. [34]). *Let \mathcal{A} and $G_n(D)$ be as in Definition 3.43. Then there is a linear function $t: \mathbb{N} \rightarrow \mathbb{N}$ so that every string of $G_n(D)$ has length at most $t(n)$.*

In other words, if $|\Sigma| = 1$ then $|G_n(D)| = O(n)$, else $|G_n(D)| = |\Sigma|^{O(n)}$.

COROLLARY 3.45. *The following structures do not have automatic presentations.*

- (i) *The free semigroup (Σ^*, \cdot) on $|\Sigma| > 1$ generators [34].*
- (ii) *Any term algebra generated by finitely many constants and at least one non-unary atomic function [34].*
- (iii) *(\mathbb{N}, \times) , multiplication of natural numbers [12].*
- (iv) *$(\mathbb{N}, \langle \cdot, \cdot \rangle)$, where $\langle \cdot, \cdot \rangle: \mathbb{N}^2 \rightarrow \mathbb{N}$ is a bijection [12].*

A straightforward application of Proposition 3.42 is the following result. It can be proved by induction on m .

PROPOSITION 3.46. *Let $(M; \cdot)$ be an automatic copy of a semigroup. Then there is a constant e , that depends only on an automaton for \cdot , so that for every $x_1, \dots, x_m \in M$,*

$$|\prod_{i=1}^m x_i| \leq \max\{|x_i| : 1 \leq i \leq m\} + e \log m.$$

This proposition will be used in Section 4 in the classification of the automatically presentable Boolean algebras and for proving certain groups do not have automatic presentations.

A different technique, independently reported by Delhommé [23] and Stephan [52], is used to prove that certain universal homogeneous structures have no automatic presentation. The basic ideas are presented below, following [35], although it is worth mentioning that Delhommé [24] presents a more general result, as well stating a corresponding condition for structures with tree-automatic presentations.

Recall $A^{\leq n}$ denotes the strings of A of length at most n .

DEFINITION 3.47. Suppose that the structure \mathcal{A} contains an atomic binary relation E and that $A \subseteq \Sigma^*$. For $n \in \mathbb{N}$ and $y \in A$ define the set $\text{tp}_n(y) \subseteq A^{\leq n}$ as those $x \in A^{\leq n}$ for which $\mathcal{A} \models E(x, y)$.

Write $\#\text{tp}_n$ for the cardinality of the set $\{\text{tp}_n(y) \mid y \in A\}$. Note that the set tp_n and the number $\#\text{tp}_n$ are defined with respect to A and E . I may write $\text{tp}_{n,A,E}$ to stress this.

In general $\#\text{tp}_n \leq |\Sigma|^{|A^{\leq n}|}$, however if \mathcal{A} is an automatic copy of some structure then we can say more.

THEOREM 3.48. *If \mathcal{A} is an automatic copy of a structure containing an atomic binary relation E , then $\#\text{tp}_{n,A,E} \leq k|A^{\leq n}|$ for some constant $k \in \mathbb{N}$ that depends on the automata for A and E .*

PROOF. One proves that there is a constant c , depending on the number of states of the automata for A and E , so that for every $n \in \mathbb{N}$ and $y \in A$, there is a $y' \in A^{\leq n+c}$ with $\text{tp}_n(y) = \text{tp}_n(y')$. Now apply the fact that $|A^{\leq n+c}| \leq k(|A^{\leq n}|)$ where the constant $k \in \mathbb{N}$ depends on the number of states of the automaton for A . \dashv

COROLLARY 3.49. *The following structures do not have automatic presentations.*

- (i) *The random graph.*
- (ii) *The universal, homogeneous partial order.*
- (iii) *The K_p -free random graph for every $p > 2$ (K_p is the complete graph on p vertices).*

PROOF. The first case is illustrated. Suppose $(A; E)$ is an automatic copy of the random graph over a binary alphabet. The random graph has the following property. For every finite subset F of A , and every partition X_1, X_2 of F , there is a vertex $x \in A$ such that $(x, x_1) \in E$ for every $x_1 \in X_1$ and $(x, x_2) \notin E$ for every $x_2 \in X_2$. Hence taking F to be $A^{\leq n}$ it holds that $\#\text{tp}_{n,A,E} = \exp(|A^{\leq n}|)$ contradicting the theorem. \dashv

We finish with another necessary condition of having an automatic presentation, due to Delhomm   [24].

DEFINITION 3.50. Say that a structure \mathcal{B} is a *sum-augmentation* of a set of structures \mathcal{S} (each having the same signature as \mathcal{B}) if there is a finite partition of $B = B_1 \cup \dots \cup B_n$ such that for each i the substructure $\mathcal{B} \upharpoonright B_i$ is isomorphic to some structure in \mathcal{S} .

THEOREM 3.51. *Suppose \mathcal{A} has finite signature. If \mathcal{A} is an automatically presentable then for every \mathcal{A} -formula $\phi(x, \bar{y})$, there is a finite set of structures \mathcal{S} so that for every tuple of elements \bar{b} from A , the substructure $\mathcal{A} \upharpoonright \phi^{\mathcal{A}}(\cdot, \bar{b})$ is a sum-augmentation of \mathcal{S} .*

PROOF. Say $\mathcal{A} = (A; R_1^{\mathcal{A}}, \dots, R_r^{\mathcal{A}})$, $A \subseteq \Sigma^*$, is an automatic copy of a structure. For any given \mathcal{A} -formula ψ , fix a deterministic automaton

$(Q_\psi, \iota_\psi, \Delta_\psi, F_\psi)$ recognising ψ^A , and write $\Gamma_\psi(w)$ for $\Delta_\psi(\iota_\psi, w)$. We will use the following property (P_ψ) : For all strings c_i, d_i with the c_i s all the same length,

$$\Delta_\psi(\Gamma_\psi(\otimes(c_1, \dots, c_k)), \otimes(d_1, \dots, d_k)) \in F_\psi.$$

if and only if $\psi(c_1d_1, \dots, c_kd_k)$ holds in \mathcal{A} .

Now, given an \mathcal{A} -formula $\phi(x, y_1, \dots, y_l)$ as in the hypothesis, and tuple \bar{b} , observe that for $m = \max\{|b_i|\}$, we can partition $\phi^A(\cdot, \bar{b})$ into the finitely many singletons $\{c\}$ for $\phi(c, \bar{b})$ with $|c| < m$, and the finitely many sets $\phi^{a\Sigma^*}(\cdot, \bar{b}) := \{aw \in A \mid \mathcal{A} \models \phi(aw, \bar{b}), w \in \Sigma^*\}$ for $|a| = m$. Since the signature is assumed to be finite, there are finitely many isomorphism types amongst substructures of the form $\mathcal{A} \upharpoonright \{a\}$, for $a \in A$. So, it is sufficient to show, that as we vary (a, \bar{b}) subject to $|a| = \max\{|b_i|\}$, there are finitely many isomorphism types amongst substructures of the form $\mathcal{A} \upharpoonright \phi^{a\Sigma^*}(\cdot, \bar{b})$.

The idea is to bound this number of isomorphism types in terms of the number of states of the automata involved. To this end, define a function f_ϕ as follows. Its domain consists of tuples (a, \bar{b}) where $|a| = \max\{|b_i|\}$; and f_ϕ sends this tuple to the tuple of states

$$(\Gamma_\phi(\otimes(a, b_1, \dots, b_l)), \Gamma_A(a), (\Gamma_{R_i^A}(\otimes(a, \dots, a)))_{i \leq r}).$$

The range of f_ϕ is bounded by $|Q_\phi| \times |Q_A| \times \prod_{i \leq r} |Q_{R_i^A}|$. In particular, the range is finite.

To finish the proof, one needs to argue that the isomorphism type of the substructure $\mathcal{A} \upharpoonright \phi^{a\Sigma^*}(\cdot, \bar{b})$ depends only on the value $f_\phi(a, \bar{b})$. This follows from the fact that if $f_\phi(a, \bar{b}) = f_\phi(a', \bar{b}')$, then the corresponding substructures are isomorphic via the mapping $I: aw \mapsto a'w$ ($w \in \Sigma^*$). Indeed, by property $(P_{x \in A})$ we get that $aw \in A$ if and only if $a'w \in A$, for every w . This means that I is a bijection between the sets $A \cap a\Sigma^*$ and $A \cap a'\Sigma^*$. Similarly, by the properties (P_ψ) where ψ is taken to be $\bar{x} \in R_i^A$, we get that I is an isomorphism between the substructures $\mathcal{A} \upharpoonright a\Sigma^*$ and $\mathcal{A} \upharpoonright a'\Sigma^*$. Finally, by (P_ϕ) we get that I is an isomorphism between the substructures $\mathcal{A} \upharpoonright \phi^{a\Sigma^*}(\cdot, \bar{b})$ and $\mathcal{A} \upharpoonright \phi^{a'\Sigma^*}(\cdot, \bar{b})$. \dashv

Here is an illustration of the theorem.

COROLLARY 3.52. [22] *The ordinal (ω^ω, \leq) is not automatically presentable.*

PROOF. Suppose for a contradiction that (ω^ω, \leq) has an automatic presentation. In Theorem 3.51, take $\phi(x, y)$ to be $x < y$ and consider the following fact (proved by induction): If the domain of a well-order, isomorphic to some ordinal of the form ω^n for $n \in \mathbb{N}$, is partitioned into finitely many pieces $\{B_i\}_i$, then there is some i so that the substructure on domain B_i is isomorphic to ω^n .

This means that the set of structures \mathcal{S} must contain $(\omega^n, <)$ for every $n \in \mathbb{N}$, contradicting the finiteness of \mathcal{S} . \dashv

3.6. Isomorphism problem. Let C be a class of structures (over a finite signature) closed under isomorphism. Write C^{aut} for the (codes of the) tuples of automata $\bar{M} = (M_A, M_-, (M_i)_i)$ in automatic presentations of members of C .

The *isomorphism problem for the automatically presentable members of C* is the set

$$\{\langle \bar{M}, \bar{M}' \rangle \mid \bar{M}, \bar{M}' \in C^{\text{aut}} \text{ present isomorphic structures}\}.$$

Note that the C^{aut} may be computable (for instance, if C are the Boolean algebras, linear orderings, or finite graphs). However this does not mean that we can describe the isomorphism types of the automatically presentable members of C^{aut} . Indeed, the isomorphism problem for the automatically presentable directed graphs is undecidable [13]:

PROOF. We encode the halting problem into this isomorphism problems as follows. For a given a Turing machine N , construct an equivalent reversible Turing machine N_r . In particular, this means that its configuration space is a disjoint union of *chains* (these are graphs isomorphic to initial segments of (\mathbb{N}, S) where $S: n \mapsto n + 1$). Such a machine can be constructed by introducing to N a tape that records the sequence of transitions that it takes, see Bennett [7].

We may assume that N (and hence also N_r) loops indefinitely instead of halting in a reject state. So, the configuration space of N_r consists of a finite chain for every accepting computation of N_r , an infinite chain for every rejecting computation of N_r , and possibly some other chains that do not correspond to valid computations of N_r (the *junk*).

The configuration space of a Turing machine N_r is automatically presentable (Example 2.4 (4)). It can be massaged so that its only finite chains are those that correspond to *valid* accepting computations of N_r . Consider the set of configurations I of N_r which have no predecessor in N_r , but are not valid initial configurations (in other words, consider the roots of chains from the junk). Write (\mathbb{N}, P) for the graph with domain \mathbb{N} and edge relation $(n+1, n)$ for every $n \in \mathbb{N}$. For each element $i \in I$, attach the chain with root i to a copy of (\mathbb{N}, P) at 0. Here *attach H to G* means take the disjoint union of H and G and add an edge from the distinguished node of g to that of h . Call the resulting graph N' . It has the property that every invalid computation is isomorphic to either (\mathbb{N}, P) (if it were a terminating computation) or to (\mathbb{Z}, S) (if it were non-terminating).

Denote by \mathcal{J} an automatic copy of the graph consisting of infinitely many disjoint copies of each of the following structures: (\mathbb{N}, S) , (\mathbb{N}, P) and (\mathbb{Z}, S) . Denote by N'' the disjoint union of the graph N' and \mathcal{J} . Then N rejects every input string if and only if N'' is isomorphic to \mathcal{J} .

It is straightforward to check that N'' is automatically presentable, and that automata presenting it can be computed from N . \dashv

In the previous proof the configuration space of N_r is *locally-finite*, namely the degree of every vertex in the underlying undirected graph is finite. The isomorphism problem for the locally-finite automatically presentable directed graphs is Π_3^0 -complete [51]. So, at first sight, the following result seems surprising.

THEOREM 3.53. [35] *The isomorphism problem for the class of automatically presentable directed graphs is Σ_1^1 -complete.*

PROOF. For hardness, we encode the isomorphism problem for computable subtrees of $\omega^{<\omega}$. A proof of its Σ_1^1 -completeness can be found in Goncharov and Knight [28, Theorem 4.4(b)] who attribute it to the folklore. Let us briefly describe this problem.

Here $\omega^{<\omega}$ is the set of all finite sequences from the set of natural numbers ω . Implicitly, there is an edge from x to $x \cdot n$ where $x \in \omega^{<\omega}$ and $n \in \omega$; this is called the *immediate successor* relation. A *subtree* of $\omega^{<\omega}$ is a subset T that is downward closed with respect to the immediate successor relation: for $x \in \omega^{<\omega}$ and $n \in \omega$, if $x \cdot n \in T$ then $x \in T$. A subtree T is *computable* if there is an algorithm that on input $x \in \omega^{<\omega}$ decides whether or not x is in T . Two subtrees are isomorphic if there is a bijection respecting the immediate successor relation. The isomorphism problem for computable subtrees of $\omega^{<\omega}$ is the set of pairs $\langle n, m \rangle$ for which n and m are indices of computable subtrees, say T_n and T_m respectively, and T_n is isomorphic to T_m .

Our aim is to exhibit, for each computable subtree T , an automatically presentable directed graph \mathcal{G}_T , so that computable subtrees T_1 and T_2 are isomorphic if and only if the directed graphs \mathcal{G}_{T_1} and \mathcal{G}_{T_2} are isomorphic. Moreover, given an index for T , we effectively construct automata presenting \mathcal{G}_T .

To begin, $\omega^{<\omega}$ has an automatic copy \mathcal{W} : the domain W is $\{\lambda\} \cup \{0, 1\}^* 1$ and immediate successor is given by $x \prec_p y \wedge \neg \exists z x \prec_p z \prec_p y$. So replacing $\omega^{<\omega}$ with \mathcal{W} , we talk instead about (*computable*) subtrees of \mathcal{W} . Now, from a computable subtree T of \mathcal{W} , construct a reversible Turing machine N_T with the property that $w \in T$ if and only if the computation of N_T starting on input w converges (compare with the previous proof). Writing \mathcal{G}'_T for the configuration space of N_T , note that \mathcal{G}'_T consists of disjoint unions of chains. Abuse notation and identify the initial configuration of \mathcal{G}'_T on input $w \in W$ with the string w .

We need an auxilliary graph \mathcal{O} formed from a copy of \mathcal{W} by attaching to every $w \in W$, infinitely many finite chains of every size and exactly one infinite chain.

Form \mathcal{G}''_T from \mathcal{G}'_T by attaching infinitely many copies of \mathcal{O} to every $w \in W \subseteq \mathcal{G}'_T$.

Finally, form the graph \mathcal{G}_T by taking the disjoint union of \mathcal{G}''_T and infinitely many chains of every size (finite and infinite). This last step takes care of the non-valid computations of N_T . It is straightforward to see that \mathcal{G}_T is automatically presentable.

For every $w \in W \subseteq G_T$,

- (i) $w \in T$ if and only if there is not an isolated infinite chain in \mathcal{G}_T whose root is an immediate successor of w (*isolated* means that every element of this chain has at most one immediate successor in \mathcal{G}_T);
- (ii) if $w \notin T$ then the tree in \mathcal{G}_T rooted at w is isomorphic to \mathcal{O} (this is because T is downward closed).

The first property ensures that every isomorphism $f : \mathcal{G}_{T_1} \cong \mathcal{G}_{T_2}$ restricts to an isomorphism between T_1 and T_2 , since f maps an isolated chain of size $\kappa \leq \omega$ to an isolated chain of the same size.

Conversely, in order to extend an isomorphism $g : T_1 \cong T_2$ to $\mathcal{G}_{T_1} \cong \mathcal{G}_{T_2}$, note that every $w \in W \subseteq G_T$ has infinitely many immediate successors $v \in W$ that are the roots in \mathcal{G}_T of trees isomorphic to \mathcal{O} . This mitigates against the case that the cardinality of $S(w) \setminus T_1$ is different from the cardinality of $S(g(w)) \setminus T_2$, where $S(\cdot) \subseteq W$ is the function denoting the set of immediate successors in the tree \mathcal{W} . \dashv

This theorem can be extended to other classes of automatically presentable structures.

If $(T; \preceq)$ is a tree, then call $(T; S_\preceq)$ a *successor tree*, where $S_\preceq(x)$ is defined as the set of immediate \preceq -successors of $x \in T$. The proof of Theorem 3.53 can easily be adapted to show that the isomorphism problem for the class of automatically presentable successor trees is Σ_1^1 -complete. We can get the result for undirected graphs by attaching a gadget (such as a cycle of length 3) to identify the root of the successor tree, and then considering the underlying graph.

Nies [46] observes that since the class of undirected graphs is bi-interpretable in a variety of other classes, one gets Σ_1^1 -completeness for these classes as well.

COROLLARY 3.54. *The isomorphism problem for each of the following classes is Σ_1^1 -complete.*

- (i) *The automatically presentable successor trees.*
- (ii) *The automatically presentable undirected graphs.*
- (iii) *The automatically presentable commutative monoids.*
- (iv) *The automatically presentable partial orders.*
- (v) *The automatically presentable lattices of height 4.*
- (vi) *The automatically presentable algebras consisting of two 1-ary functions.*

§4. Classifications. Fix a class C of structures (say the class of Boolean algebras, or linear orders). Which members of C are automatically presentable? This section is concerned with describing, in terms of classical invariants, the isomorphism types of the automatic members of C .

Each subsection begins with the classification of the unary-automatically presentable structures in the given class. These are usually based on the

following proposition, whose proof follows from an analysis of the structure of the corresponding automata.

PROPOSITION 4.1. [9, 51] *If a graph $\mathcal{G} = (G; E)$ is a unary-automatically presentable, then it contains a finite number of infinite connected components, and a finite bound on the sizes of the finite connected components.*

For the non-unary case, Proposition 2.6 says that it is sufficient to consider $|\Sigma| = 2$.

Remark 4.2. Each of the following subsections indicate that the classification of classes of unary-automatically presentable structures is considerably simpler than the classification of classes of automatically presentable structures over a binary alphabet.

4.1. Equivalence structures. An equivalence structure $(E; \rho)$ is one where ρ is an equivalence relation on the set E . Each equivalence structure is characterised up to isomorphism by the number of equivalence classes of every size. To this end, define the *height function* $h_{\mathcal{E}}: \mathbb{N} \cup \{\infty\} \rightarrow \mathbb{N} \cup \{\infty\}$ of an equivalence structure \mathcal{E} as $h(n)$ being the number (possibly infinite) of equivalence classes of size n (possibly infinite). Then \mathcal{E}_1 is isomorphic to \mathcal{E}_2 if and only if $h_{\mathcal{E}_1} = h_{\mathcal{E}_2}$.

The unary case follows immediately from Proposition 4.1.

THEOREM 4.3. [9, 51] *An equivalence structure $(E; \rho)$ has a unary-automatic presentation if and only if $h_{\mathcal{E}}(\infty)$ is finite and $h_{\mathcal{E}}(n) = 0$ for all but finitely many $n \in \mathbb{N}$.*

We now turn to the non-unary case. Consider an automatically presentable equivalence structure \mathcal{E} . The set of elements of E in infinite classes is definable using \exists^∞ , and so we can compute how many infinite classes there are. So to characterise the automatically presentable equivalence structures it is sufficient to consider those with no infinite classes. To this end, write \mathcal{H} for the set of height functions of automatically presentable equivalence structures with no infinite classes. Adopt the convention that $m + n = m \times n = \infty$ if at least one of m or n is ∞ .

PROPOSITION 4.4. [51]

- (i) \mathcal{H} contains all functions of the form $h_f: \mathbb{N} \rightarrow \mathbb{N}$ defined by $h(f(n)) = 1$, where $f: \mathbb{N} \rightarrow \mathbb{N}$ is either a polynomial (with coefficients in \mathbb{N}) or an exponential k^{an+b} (for some $k, a, b \in \mathbb{N}$).
- (ii) If $h, h' \in \mathcal{H}$ then so is their
 - (a) sum $(h + h')(n) = h(n) + h'(n)$,
 - (b) Dirichlet convolution $(h * h')(n) = \sum_{ab=n} h(a)h'(b)$, and
 - (c) Cauchy product $(h \# h')(n) = \sum_{a+b=n} h(a)h'(b)$.

PROOF. For the first item, consider automatic copy $(R; \text{el})$ of an equivalence structure where $R \subseteq \Sigma^*$ and el is the equal length predicate. Thus the equivalence classes are of the form $R^{=n}$ for $n \in \mathbb{N}$. But for every polynomial

or exponential function f as in the statement of the proposition, there is a regular set $R_f \subseteq \Sigma^*$ with $f(n) = |(R_f)^{=n}|$ (see [53] or [51, Lemma D.2.3]).

For the second item, sum corresponds to disjoint union of equivalence structures, and Dirichlet convolution corresponds to direct product. Cauchy product is only slightly more involved. \dashv

So in particular there is an automatically presentable equivalence structure whose height function has unbounded range. As far as I know, neither is there a classification of \mathcal{H} , nor is it known whether the isomorphism problem for automatically presentable equivalence relations, seen to be Π_1^0 , is decidable.

4.2. Linear orders. An excellent reference for linear orders is [50]. The classical ranking of linear orders $\mathcal{L} = (L; \leq)$ is based on iteratively factoring \mathcal{L} by the equivalence relation c stating that x is equivalent to y if the number of elements between x and y is finite.

For ease of reference, here is the formal definition. For every ordinal α define an equivalence relation \sim_α on \mathcal{L} inductively: $x \sim_\alpha y$ if $x = y$ or for some $\beta < \alpha$, the number of elements in \mathcal{L}/\sim_β between $[x]_{\sim_\beta}$ and $[y]_{\sim_\beta}$ is finite. Here \mathcal{L}/\sim_β is the linear ordering defined as follows: its domain is the collection of non-empty \sim_β -equivalence classes $[x]_{\sim_\beta}$ ($x \in L$). The order is defined by $[x]_{\sim_\beta}$ less than $[y]_{\sim_\beta}$ if $a <_L b$ for every $a \in [x]_{\sim_\beta}$, $b \in [y]_{\sim_\beta}$.

The FC-rank of \mathcal{L} is defined as the least ordinal α so that for every $\beta < \alpha$ and every $x \in L$, we have $[x]_{\sim_\beta} = [x]_{\sim_\alpha}$. For example, the FC-rank of the ordinal ω^α is α .

Here FC is an acronym for ‘finite condensation’, hinting that elements a finite distance apart are condensed together.

A linear order \mathcal{L} is *dense* if for every distinct $a, b \in L$, there is a $z \in L$ with $a < z < b$. Note that the linear order with exactly one element is dense. A linear order is *scattered* if none of its suborderings are both dense and infinite. The *ordered sum* of the orderings \mathcal{A}_b indexed by the linear order \mathcal{B} is the result of replacing each $b \in \mathcal{B}$ by a copy of \mathcal{A}_b , and is written $\Sigma_{\mathcal{B}} \mathcal{A}_b$.

Fact 4.5. Every linear order \mathcal{L} is a dense sum of scattered linear orders. That is, \mathcal{L} can be expressed as $\Sigma_{\mathcal{D}} \mathcal{L}_d$ for some dense \mathcal{D} and scattered \mathcal{L}_d s, $d \in \mathcal{D}$.

I do not know of a non-machine theoretic classification of the automatically presentable linear orders, or even the FC-rank 1 linear orders (it is straightforward to exhibit a linear order of FC-rank 1 with undecidable first-order theory). The next result describes the unary case.

THEOREM 4.6. [9, 51] *A linear order $(L; \leq)$ has a unary-automatic presentation if and only if it is a finite sum of scattered orders of FC-rank at most 1.*

In other words, the linear orders with unary-automatic presentations are finite sums of linear orders amongst the set ω , ω^* , and \mathbf{n} , for $n < \omega$. In

particular, the order type of the rationals is not automatically presentable over a unary alphabet, and the least ordinal without a unary automatic presentation is ω^2 .

Regarding the general (non-unary) case, Khoussainov and Nerode asked for the least ordinal without an automatic presentation [34] (it is easy to see that the automatically presentable ordinals form an initial segment of the countable ordinals). In Corollary 3.52, we saw that the condition of Theorem 3.51 implies that the answer is ω^ω .

Similarly, we can use the condition of Theorem 3.51 to get a generalisation to linear orders [37].

THEOREM 4.7. *The FC-rank of every automatically presentable linear order is finite.*

PROOF. In Theorem 3.51, let \mathcal{L} be an automatically presentable linear order, and $\phi(x, y_1, y_2)$ be the relation $y_1 \leq x \leq y_2$. Then there is a finite set of structures \mathcal{S} , so that for every $a_1 \leq a_2 \in L$, there is a partition of the domain of $[a_1, a_2]$ into finitely many pieces $\{A_i\}$, so that every $\mathcal{L} \upharpoonright A_i$ is isomorphic to some structure in \mathcal{S} .

Just as in Corollary 3.52, we look for a decomposition property such as: If the domain of an automatically presentable linear order \mathcal{L} is partitioned into finitely many pieces $\{B_i\}_i$, then some $\mathcal{L} \upharpoonright B_i$ has the same FC-rank as \mathcal{L} . Of course this condition fails horribly if \mathcal{L} is not scattered, because \mathcal{L} then embeds orders of arbitrary countable FC-rank. However, in the scattered case it just falls short; for instance, partition $\mathcal{L} = \omega + \omega$ into the first copy of ω ($B_1 = \{0, 1, \dots\}$) and the second copy ($B_2 = \{\omega, \omega + 1, \dots\}$). Then $\text{FC}(\mathcal{L}) = 2$, but $\text{FC}(B_i) = 1$.

By slightly altering the notion of rank we can achieve the required decomposition result for scattered linear orders. Define the FC_* -rank of a scattered linear order \mathcal{L} as the least ordinal α such that \mathcal{L} can be expressed as a finite sum of orders of FC-rank at most α . Then $\text{FC}_*(\mathcal{L}) \leq \text{FC}(\mathcal{L}) \leq \text{FC}_*(\mathcal{L}) + 1$.

The following decomposition property can be proved by induction on FC_* -rank: If the domain of a scattered linear order \mathcal{L} is partitioned into finitely many pieces $\{B_i\}_i$, then there is some i with $\text{FC}_*(\mathcal{L} \upharpoonright B_i) = \text{FC}_*(\mathcal{L})$.

Combining this with the statement in the first paragraph, we see that for every $a_1 \leq a_2 \in L$, if $\mathcal{L} \upharpoonright [a_1, a_2]$ is scattered, say with FC_* -rank α , then \mathcal{S} contains a scattered linear order of FC_* -rank α . Moreover, α is finite; for otherwise, using elementary properties of rank, $\mathcal{L} \upharpoonright [a_1, a_2]$ would contain infinitely many closed scattered subintervals having pairwise distinct FC_* -ranks, contradicting the finiteness of \mathcal{S} . Thus there is a uniform finite bound, say k , on the FC-rank of every closed scattered subinterval of \mathcal{L} .

Now if \mathcal{L} is scattered, then every two elements $a_1, a_2 \in L$ are condensed within k steps; thus $\text{FC}(\mathcal{L}) \leq k$. Incase \mathcal{L} is not scattered, by Fact 4.5, \mathcal{L} is the sum of scattered orders \mathcal{L}_d ($d \in \mathcal{D}$, \mathcal{D} dense). Consequently, each \mathcal{L}_d has FC-rank at most k , and so \mathcal{L} has FC-rank at most k . \dashv

Some decidability results for automatically presentable linear orders now follow. Since the axioms stating that \leq linearly orders L are first-order, it is decidable whether a given automatically presentable structure $(L; \leq)$ is a linear order or not.

COROLLARY 4.8. [37] *Let $\mathcal{L} = (L; \leq)$ be an automatically presentable linear order.*

- (i) *It is decidable whether or not \mathcal{L} is scattered. In case \mathcal{L} is not scattered, we can effectively compute an automatically presentable dense subordering \mathcal{D} and automatically presentable scattered suborderings \mathcal{L}_d for which $\mathcal{L} = \Sigma_{\mathcal{D}} \mathcal{L}_d$.*
- (ii) *It is decidable whether or not \mathcal{L} is isomorphic to an ordinal.*
- (iii) *The isomorphism problem for automatically presentable ordinals is decidable. In fact the Cantor-normal-form may be extracted from an automatic copy of an ordinal.*

It is not known whether the isomorphism problem for automatically presentable linear orderings is decidable.

Cantor's theorems. One of Cantor's theorems says that every countable linear ordering embeds in the rational ordering \mathbb{Q} .

There are, potentially, a variety of possible automatic versions. The following proposition is the best known.

PROPOSITION 4.9. [38] *Every automatic copy \mathcal{M} of a linear order can be embedded into some automatic copy of \mathbb{Q} by a function $f : \mathcal{M} \rightarrow \mathbb{Q}$ with the following properties:*

- (i) *The function f is continuous with respect to the order topology.*
- (ii) *The graph of f is regular.*

PROOF. We mainly present the definition of the required automatic copy of \mathbb{Q} . Let $\mathcal{M} = (M; \leq_M)$ be an automatic copy of a linear order, where $M \subseteq \Delta^*$ and $0, 1 \notin \Delta$. Define $M_L \subseteq M$ as the set of elements of \mathcal{M} that cannot be approximated from the left. That is, $w \in M_L$ if and only if

$$(\exists u)[u <_M w \wedge \neg(\exists z)(u <_M z <_M w)] \vee \neg(\exists u)[u <_M w].$$

Similarly define $M_R \subseteq M$ as the set of elements that cannot be approximated from the right. The required automatic copy of \mathbb{Q} has domain

$$M \cup M_L \cdot 0\{0, 1\}^* \cup M_R \cdot 1\{0, 1\}^*$$

and relation $uw \leq u'w'$ (here $u, u' \in M$ and $w, w' \in \{0, 1\}^*$) if $u \leq_M u'$ or otherwise $u = u'$ and $w \sqsubseteq_Q w'$. Here $(\{0, 1\}^*, \sqsubseteq_Q)$ is the automatic copy of \mathbb{Q} from Example 2.4 (5). The required function f maps $u \mapsto u$ for $u \in M$. \dashv

It is not known whether there is a single automatic copy of \mathbb{Q} that embeds, in the sense above, all automatic copies of all automatically presentable linear orders \mathcal{M} .

Cantor also proved that \mathbb{Q} is homogeneous: For every two tuples $x_1 < \dots < x_m$ and $y_1 < \dots < y_m$ there is an automorphism $f: \mathbb{Q} \rightarrow \mathbb{Q}$ with $f(x_i) = y_i$ for $i \leq m$. Again there might be a number of automatic variations. Call an automatic copy of \mathbb{Q} *automatically homogeneous* if for every two tuples there is an automorphism as above that is also regular.

PROPOSITION 4.10. [38] *The automatic copy of \mathbb{Q} in Example 2.4(5) is automatically homogeneous. There exists an automatic copy of \mathbb{Q} that is not automatically homogeneous.*

4.3. Trees. A tree $(T; \preceq)$ is a partial order with a smallest element and the property that for every element $v \in T$, the set $\{w \in T \mid w \preceq v\}$ is finite and linearly ordered by \preceq .

For $u \in T$, write $S(u)$ for the set of \prec -immediate successors of u ; namely,

$$S(u) = \{v \in T \mid u \prec v \wedge \forall z[u \prec z \preceq v \rightarrow z = v]\}.$$

A tree is *finitely-branching* if $S(u)$ is finite for every $u \in T$. An *infinite path* in a tree is a sequence of elements $(w_i)_{i \in \mathbb{N}}$ with $w_{i+1} \in S(w_i)$ for every i .

The complexity of the isomorphism problem for automatically presentable trees, over the signature of partial orders, is not known. However, another measure of the complexity of a tree is its Cantor–Bendixson rank. Before giving the definition, we need some preliminary concepts. Define the extendible part $E(\mathcal{T})$ of \mathcal{T} as those $x \in T$ that are on some infinite path of T . Define $d(\mathcal{T})$, the *derivative* of \mathcal{T} , as the subtree of \mathcal{T} restricted to those elements that are on two distinct infinite paths, namely:

$$\{x \in T \mid \exists z, z' \in E(\mathcal{T}), z, z' \succ x \text{ and neither } z \preceq z' \text{ nor } z' \preceq z\}.$$

For each ordinal α define the iterated operation $d^\alpha(\mathcal{T})$ inductively by

$$d^\alpha(\mathcal{T}) = \{x \in T : \forall \beta < \alpha [x \text{ is an element of the tree } d(d^\beta(\mathcal{T}))]\}.$$

Note that for $\alpha = 0$ the range of the universal quantifier is void, and therefore $d^0(\mathcal{T})$ is just T .

DEFINITION 4.11. The *Cantor–Bendixson rank* $\text{CB}(\mathcal{T})$ of a tree \mathcal{T} is the least ordinal α such that $d^\alpha(\mathcal{T}) = d^{\alpha+1}(\mathcal{T})$.

Since \mathcal{T} is countable, $\text{CB}(\mathcal{T})$ is a countable ordinal. Also, if \mathcal{T} contains countably many infinite paths, then $d^{\text{CB}(\mathcal{T})}$ is the empty tree.

THEOREM 4.12. [37] *The CB-rank of every automatically presentable tree $\mathcal{T} = (T; \preceq)$ is finite and computable from \mathcal{T} .*

The main idea of the proof is to associate to each automatically presentable tree \mathcal{T} an automatically presentable linear ordering, the Kleene–Brouwer ordering for instance, whose FC-rank can be used to bound the CB-rank of \mathcal{T} . By Theorem 4.7 this FC-rank is finite.

König's Lemma. König's Lemma says that an infinite finitely-branching tree has an infinite path. Here is an automatic analogue.

THEOREM 4.13. *If $\mathcal{T} = (T; \preceq)$ is an automatic copy of an infinite finitely-branching tree, then \mathcal{T} has a regular infinite path. That is, there exists a regular set $P \subseteq T$ where P is an infinite path of \mathcal{T} .*

PROOF. Define a set P as those elements x such that $\exists^\infty w[x \prec w]$ and for which every $y \prec x$ satisfies that

$$\forall z, z' \in S(y)[z \preceq x \Rightarrow z \leq_{\text{lex}} z'].$$

Then P is the length-lexicographically least infinite path of \mathcal{T} (in the ordering induced by the finite strings presenting the tree). \dashv

However, using the 2-Ramsey quantifier we can do more.

THEOREM 4.14. *If \mathcal{T} is an automatic copy of a tree with countably many infinite paths, then every infinite path is regular.*

PROOF. Denote by $E(\mathcal{T}) \subseteq T$ the set of elements of a tree \mathcal{T} that are on infinite paths. It is definable in \mathcal{T} using the 2-Ramsey quantifier, so Theorem 3.20 gives that $E(\mathcal{T})$ is regular. Then every isolated path of \mathcal{T} is regular, since it is definable as $\{x \in E(\mathcal{T}) \mid p \preceq x\} \cup \{x \in E(\mathcal{T}) \mid x \prec p\}$, for suitable $p \in E(\mathcal{T})$. Replace \mathcal{T} by its derivative $d(\mathcal{T})$, which is also automatically presentable. Since the CB-rank of \mathcal{T} is finite (Theorem 4.12), and $d^{\text{CB}(\mathcal{T})}$ is the empty tree, every infinite path is defined in this way. \dashv

The trees considered above are partial orders. Instead we may consider successor trees: structures of the form $(T; S, r)$ where S is the immediate successor relation and r is the root. These behave quite differently from trees in the signature of partial orders. By Theorem 3.53, the isomorphism problem for successor trees is Σ_1^1 -complete. A slight modification of Corollary 3.41 yields the following pathology.

PROPOSITION 4.15. *There is an automatic copy $(T; S, r)$ of a successor tree with exactly two infinite paths, neither of which is a regular subset of T .*

4.4. Boolean algebras. Boolean algebras are considered in the signature (\vee, \wedge, \neg) . Write \emptyset for the bottom element, 1 for the top and \subseteq for the associated partial order.

The following lemma, which will be useful for classifying the automatically presentable Boolean algebras, is a direct application of Proposition 3.46.

LEMMA 4.16. *Suppose \mathcal{B} is an automatic copy of a Boolean algebra. There exists a constant $e \in \mathbb{N}$, that depends only on the automaton recognising \vee , such that for every set S of n elements of \mathcal{B} , the length of the element $\bigvee S$ is at most*

$$\max_{s \in S}\{|s|\} + e \log n.$$

PROPOSITION 4.17. *The countable atomless Boolean algebra is not automatically presentable.*

PROOF. Suppose $\mathcal{B} = (B; \vee, \wedge, \neg)$ is an automatic copy of the countable atomless Boolean algebra. The idea is that, starting with the top element I , we can generate too many pairwise disjoint elements by repeated splitting.

For non-zero $x \in B$, there is exists a non-zero element $a < x$, because x is not an atom. Thus there are two disjoint non-zero elements $(x \wedge a)$ and $(x \wedge \neg a)$ below x . Define $a(x)$ as the length-lexicographically least element a with this property. Expand the presentation of \mathcal{B} to include the functions $f_l: x \mapsto x \wedge a(x)$ and $f_r: x \mapsto x \wedge \neg a(x)$. Apply Proposition 3.44 with respect to the functions f_l, f_r and with $D = \{I\}$: There exists a linear function t so that every string in $G_n(I)$ has length at most $t(n)$.

The subalgebra \mathcal{A} of \mathcal{B} generated by $G_n(\{I\})$ has $\exp(n)$ atoms. By Lemma 4.16, there is a constant e , so that every element of \mathcal{A} has length at most $t(n) + en$. So, there are only $|\Sigma|^{t(n)+en}$ available strings to code for elements of \mathcal{A} ; contradicting the fact that \mathcal{A} has $\exp_2(n)$ elements. \dashv

Consequently, no automatically presentable Boolean algebra has the property that the countable atomless boolean algebra is definable in it. In particular, if an automatically presentable Boolean algebra has finitely many atoms, then it is finite.

COROLLARY 4.18. *There are no infinite unary-automatically presentable Boolean algebras.*

PROOF. Suppose \mathcal{B} is a unary-automatic copy of a Boolean algebra with infinitely many atoms. The set of atoms B_A , being first-order definable in \mathcal{B} , is regular. Apply Proposition 3.44 to the functions \vee, \wedge , and \neg , and the set B_A . For $n \geq 3$, the set $G_{n+2}(B_A)$ contains every element generated by the first n atoms. So $|G_{n+2}(B_A)| \geq 2^n$, which exceeds the linear bound. \dashv

Recall that \mathcal{B}_{fin} is the Boolean algebra of finite or co-finite subsets of \mathbb{N} . Write $\mathcal{B}_{\text{fin}}^n$ for the n -fold power. A refinement of the proof of Proposition 4.17 is used to classify the automatically presentable Boolean algebras in the non-unary case.

THEOREM 4.19. [35] *An infinite Boolean algebra is automatically presentable if and only if it is isomorphic to $\mathcal{B}_{\text{fin}}^n$ for some $n \in \mathbb{N}$.*

COROLLARY 4.20. *The isomorphism problem for automatically presentable Boolean algebras is decidable.*

PROOF. Start with an automatic copy of an infinite Boolean algebra \mathcal{B} . Recall that the Frechét congruence on \mathcal{B} consists of those pairs whose symmetric difference is either 0 or a meet of a finite number of atoms of \mathcal{B} . This congruence is first-order definable in \mathcal{B} with the additional quantifier \exists^∞ , and so is regular. Hence the quotient of \mathcal{B} by its Frechét congruence is automatically presentable. Compute the least number of times one has to factor \mathcal{B} until one reaches the two element algebra, say n times. This value defines \mathcal{B} up to isomorphism; indeed, \mathcal{B} is isomorphic to $\mathcal{B}_{\text{fin}}^n$. \dashv

4.5. Groups, rings and fields. By algebraically characterising the relations recognised by automata over a unary alphabet, Blumensath establishes the following theorem.

THEOREM 4.21. [9] *There are no infinite unary-automatically presentable groups $(G; \cdot)$, rings, or fields.*

Let us consider the non-unary cases. Recall that an integral domain $(D; +, \mathbf{0}, \cdot, \mathbf{1})$ is a commutative ring with identity with the property that if $x \cdot y = \mathbf{0}$ then $x = \mathbf{0}$ or $y = \mathbf{0}$. With a little work, the ideas in Section 3.5 (Restriction on growth) can be used to prove the next result.

THEOREM 4.22. [35] *There are no infinite automatically presentable integral domains. In particular, there are no infinite automatically presentable fields.*

However, the cases of groups and rings are open. The current state of affairs is nicely detailed in Nies [46]. I only mention a sample here.

For $k \geq 2$, write $\mathbb{Z}[1/k]$ for the additive group of rationals of the form z/k^i , where $z \in \mathbb{Z}$ and $i \in \mathbb{N}$. It is straightforward to show that $\mathbb{Z}[1/k]$ and $\mathbb{Z}[1/k]/\mathbb{Z}$ have automatic presentations.

PROPOSITION 4.23. [35] *The following groups do not have automatic presentations:*

- (i) *The positive rationals under multiplication.*
- (ii) *The direct sum of infinitely many copies of $\mathbb{Z}[1/k]$, for a fixed k .*
- (iii) *The direct sum of infinitely many copies of the Prüfer group $\mathbb{Z}[1/k]/\mathbb{Z}$, for a fixed k .*

PROOF. The idea, illustrated here in the proof of the first item, is to define sets F_n of lots of distinct elements using the primes of length at most n , and then show, as before using Proposition 3.46, that the lengths of these elements are too ‘short’.

Suppose that (Q^+, \times) is automatically presentable; say an automatic copy is $(D; \times)$. Let P be the elements of D that correspond to primes. Recall $P^{\leq n}$ are the elements of P of length at most n . Let

$$F_n = \left\{ \prod_{p \in P^{\leq n}} p^{\beta_p} : 0 \leq \beta_p \leq 2^n \right\}.$$

Then F_n contains $\exp(nr_n)$ distinct elements, where $r_n = |P^{\leq n}|$.

On the other hand, we compute an upperbound on the size of F_n . By successive applications of Proposition 3.46, there is a constant e , so that $|p^\beta| \leq n + en \leq n(e+1)$ and so each string has length at most $n(e+1) + e \log r_n$. This places an upper bound of $\exp(n(e+1) + e \log r_n)$ on the size of F_n , contradicting that r_n goes to infinity. \dashv

Using a mixture of algebra, coding techniques, and growth arguments, Nies and Thomas [47] prove the next result (the first item was already established for finitely generated groups \mathcal{G} by Oliver and Thomas [48]).

- THEOREM 4.24.** (i) *Let \mathcal{G} be an automatically presentable infinite group. Then every finitely generated subgroup of \mathcal{G} has an Abelian subgroup of finite index.*
(ii) *Every finitely generated subring of an automatically presentable ring is finite.*

§5. Open problems. This work has only dealt with automata operating on finite strings. The fundamental result that makes the theory work is Theorem 3.1 [Definability]. Analogues of this theorem hold for other models of finite automata: automata operating on finite trees (ranked or unranked), infinite strings, and infinite trees. Each model yields a class of ‘automatically presentable structures’ where the basic theory goes through: particularly, a logical characterisation via interpretability similar to Theorem 3.8. Of course, the problems considered in this survey—such as classification, intrinsic regularity, and proving non-automaticity—can be asked in these more general settings, see for instance [24, 39, 21].

However, there are still many problems in the finite string case. Here are some problems whose solutions will likely require new techniques.

Problem 5.1. Find new ways to prove that a structure does not have an automatic presentation. For instance, do the following structures have automatic presentations?

- (i) The additive group of rationals $(\mathbb{Q}; +)$. This structure is WMSO-interpretable in a certain infinite string (compare [46]). This limits the kind of proof that could be used to show that $(\mathbb{Q}; +)$ has no finite-string automatic presentation.
- (ii) The graph generated by the ground term rewriting system with one constant a , one binary function f , initial term $f(a, a)$ and rewriting rule $a \mapsto f(a, a)$. In other words, the structure whose domain consists of all finite binary branching trees t , and for which there is an edge from t to t' if t' extends t by exactly one node. This is a candidate for separating the class of graphs generated by ground term rewriting systems from the finite-string automatic graphs (see [42]).

Problem 5.2. Investigate the complexity in the arithmetic/analytic hierarchy of the isomorphism problem for classes of automatically presentable structures. For instance, what is the complexity of the isomorphism problem for finite-string automatically presentable linear orders, or equivalence structures?

Problem 5.3. Can we capture intrinsic regularity using definability?

For instance, is it the case that for every finite-string automatically presentable structure \mathcal{A} ,

$$\text{FO}[\cup_n Q_n^{\text{reg}}]_{\omega-\text{inv}}(\mathcal{A}) = \text{IR}(\mathcal{A})?$$

REFERENCES

- [1] C. J. ASH and A. NERODE, *Intrinsically recursive relations*, *Aspects of effective algebra (clayton, 1979)*, Upside Down A Book Co., Yarra Glen, 1981, pp. 26–41.
- [2] V. BÁRÁNY, *Invariants of automatic presentations and semi-synchronous transductions*, *STACS 2006* (B. Durand and W. Thomas, editors), Lecture Notes in Computer Science, vol. 3884, Springer, 2006, pp. 289–300.
- [3] ———, *Automatic presentations of infinite structures*, Ph.D. thesis, RWTH Aachen, 2007.
- [4] V. BÁRÁNY, L. KAISER, and S. RUBIN, *Countable omega-automatic structures and their presentations*, *STACS 2008* (S. Albers, P. Weil, and C. Rochange, editors), Dagstuhl Seminar Proceedings, vol. 08001, Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany, 2008.
- [5] M. BENEDIKT and L. LIBKIN, *Tree extension algebras: logics, automata, and query languages*, *LICS 2002*, IEEE Computer Society, 2002, pp. 203–212.
- [6] M. BENEDIKT, L. LIBKIN, T. SCHWENTICK, and L. SEGOUFIN, *A model-theoretic approach to regular string relations*, *LICS 2001*, IEEE Computer Society, 2001, pp. 431–440.
- [7] C. H. BENNETT, *Logical reversibility of computation*, *IBM Journal of Research and Development*, vol. 17 (1973), no. 6, pp. 525–532.
- [8] JEAN BERSTEL, *Transductions and context-free languages*, Leitfäden der Angewandten Mathematik und Mechanik [Guides to Applied Mathematics and Mechanics], vol. 38, B. G. Teubner, Stuttgart, 1979.
- [9] A. BLUMENSATH, *Automatic structures*, Diploma thesis, RWTH Aachen, 1999.
- [10] ———, *Prefix-recognisable graphs and monadic second-order logic*, Technical report, RWTH Aachen, 2001.
- [11] ———, *Axiomatising tree-interpretable structures*, *STACS 2002* (H. Alt and A. Ferreira, editors), Lecture Notes in Computer Science, vol. 2285, Springer, 2002, pp. 596–607.
- [12] A. BLUMENSATH and E. GRÄDEL, *Automatic structures*, *LICS 2000*, IEEE Computer Society, 2000, pp. 51–62.
- [13] ———, *Finite presentations of infinite structures: Automata and interpretations*, *Proceedings of the 2nd International Workshop on Complexity in Automated Deduction, CiAD 2002*, 2002.
- [14] ———, *Finite presentations of infinite structures: Automata and interpretations*, *Theory of Computing Systems*, vol. 37 (2004), pp. 641–674.
- [15] V. BRUYÈRE, G. HANSEL, C. MICHAUX CHRISTIAN, and R. VILLEMAIRE, *Logic and p-recognizable sets of integers*, *Bulletin of the Belgian Mathematical Society. Simon Stevin*, vol. 1 (1994), no. 2, pp. 191–238, Journées Montois (Mons, 1992).
- [16] J. R. BÜCHI, *Weak second-order arithmetic and finite automata*, *Zeitschrift für mathematische Logik und Grundlagen der Mathematik*, vol. 6 (1960), pp. 66–92.
- [17] ———, *On a decision method in restricted second order arithmetic*, *Logic, methodology and philosophy of science*, Stanford University Press, 1962, pp. 1–11.
- [18] J. W. CANNON, *The combinatorial structure of cocompact discrete hyperbolic groups*, *Geometriae Dedicata (Historical Archive)*, vol. 16 (1984), no. 2, pp. 123–148.
- [19] J. W. CANNON, D. B. H. EPSTEIN, D. F. HOLT, S. V. F. LEVY, M. S. PATERSON, and W. P. THURSTON, *Word processing in groups*, Jones and Bartlett, 1992.
- [20] D. CENZER and J. B. REMMEL, *Complexity-theoretic model theory and algebra*, *Handbook of recursive mathematics, vol. 1* (Yu. L. Ershov, S. S. Goncharov, A. Nerode, and J. B. Remmel, editors), Studies in Logic and the Foundations of Mathematics, vol. 138, North-Holland, Amsterdam, 1998, pp. 381–513.
- [21] T. COLCOMBET and C. LÖDING, *Transforming structures by set interpretations*, Technical Report AIB-2006-07, RWTH Aachen, 2006.

- [22] C. DELHOMMÉ, *Non-automaticity of ω^ω* , 2001, manuscript.
- [23] ———, *The Rado graph is not automatic*, 2001, manuscript.
- [24] ———, *Automaticité des ordinaux et des graphes homogènes*, *Comptes Rendus Mathématique*, vol. 339 (2004), no. 1, pp. 5–10.
- [25] J. DONER, *Tree acceptors and some of their applications*, *Journal of Computer and System Sciences*, vol. 4 (1970), pp. 406–451.
- [26] S. EILENBERG, C. C. ELGOT, and J. C. SHEPHERDSON, *Sets recognised by n -tape automata*, *Journal of Algebra*, vol. 13 (1969), no. 4, pp. 447–464.
- [27] C. C. ELGOT, *Decision problems of finite automata design and related arithmetics*, *Transactions of the American Mathematical Society*, vol. 98 (1961), pp. 21–51.
- [28] S. S. GONCHAROV and J. F. KNIGHT, *Computable structure and non-structure theorems*, *Algebra and Logic*, vol. 41 (2002), no. 6, pp. 351–373.
- [29] L. HELLA, *Definability hierarchies of generalized quantifiers*, *Annals of Pure and Applied Logic*, vol. 43 (1989), no. 3, pp. 235–271.
- [30] B. R. HODGSON, *Théories décidables par automate fini*, Ph.D. thesis, University of Montréal, 1976.
- [31] ———, *On direct products of automaton decidable theories*, *Theoretical Computer Science*, vol. 19 (1982), pp. 331–335.
- [32] ———, *Decidabilite par automate fini*, *Annales de Sciences Mathématiques du Québec*, vol. 7 (1983), pp. 39–57.
- [33] D. F. HOLT, *The Warwick automatic groups software*, *Geometric and computational perspectives on infinite groups (Minneapolis, MN and New Brunswick, NJ, 1994)*, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol. 25, American Mathematical Society, 1996, pp. 69–82.
- [34] B. KHOUSSAINOV and A. NERODE, *Automatic presentations of structures*, Lecture Notes in Computer Science, vol. 960, 1995.
- [35] B. KHOUSSAINOV, A. NIES, S. RUBIN, and F. STEPHAN, *Automatic structures: Richness and limitations*, *Logical Methods in Computer Science*, vol. 3 (2007), no. 2, pp. 0–0.
- [36] B. KHOUSSAINOV, S. RUBIN, and F. STEPHAN, *Definability and regularity in automatic structures*, *STACS 2004* (V. Diekert and M. Habib, editors), Lecture Notes in Computer Science, vol. 2996, Springer, 2004, pp. 440–451.
- [37] ———, *Automatic linear orders and trees*, *ACM Transactions on Computational Logic*, vol. 6 (2005), no. 4, pp. 675–700.
- [38] D. KUSKE, *Is Cantor’s theorem automatic?*, *LPAR 2003* (M. Y. Vardi and A. Voronkov, editors), Lecture Notes in Artificial Intelligence, vol. 2850, Springer, 2003, pp. 332–345.
- [39] D. KUSKE and M. LOHREY, *First-order and counting theories of omega-automatic structures*, Fakultätsbericht Nr. 2005/07, Universität Stuttgart, Fakultät Informatik, Elektrotechnik und Informationstechnik, September 2005.
- [40] L. LIBKIN and F. NEVEN, *Logical definability and query languages over unranked trees*, *LICS 2003*, IEEE Computer Society, 2003, pp. 178–187.
- [41] PER LINDSTRÖM, *First order predicate logic with generalized quantifiers*, *Theoria*, vol. 32 (1966), pp. 186–195.
- [42] CHRISTOF LÖDING, *Infinite graphs generated by tree rewriting*, Ph.D. thesis, RWTH Aachen, 2003.
- [43] M. LOHREY, *Automatic structures of bounded degree*, *LPAR 2003* (M. Y. Vardi and A. Voronkov, editors), Lecture Notes in Artificial Intelligence, vol. 2850, Springer, 2003, pp. 344–358.
- [44] C. MICHAUX and F. POINT, *Les ensembles k -reconnaissables sont définissables dans $\langle N, +, V_k \rangle$* , *Comptes Rendus des Séances de l’Académie des Sciences. Série I. Mathématique*, vol. 303 (1986), no. 19, pp. 939–942.

- [45] A. A. NABEBIN, *Multitape automata in a unary alphabet*, *Trudy Moskovskogo Ordena Lenina Ènergeticheskogo Instituta*, vol. 292 (1976), pp. 7–11.
- [46] A. NIES, *Describing Groups*, this BULLETIN, vol. 13 (2007), no. 3, pp. 305–339.
- [47] A. NIES and R. THOMAS, *FA-presentable groups and rings*, 2005, to appear.
- [48] G. OLIVER and R. THOMAS, *Automatic presentations of finitely generated groups*, *STACS 2005* (V. Diekert and B. Durand, editors), Lecture Notes in Computer Science, vol. 3404, Springer, 2005.
- [49] M. O. RABIN, *Decidability of second-order theories and automata on infinite trees*, *Transactions of the American Mathematical Society*, vol. 141 (1969), pp. 1–35.
- [50] J. G. ROSENSTEIN, *Linear orderings*, Academic Press, 1982.
- [51] S. RUBIN, *Automatic structures*, Ph.D. thesis, University of Auckland, 2004.
- [52] F. STEPHAN, *The random graph is not automatically presentable*, 2002, manuscript.
- [53] A. SZILARD, S. YU, K. ZHANG, and J. SHALLIT, *Characterizing regular languages with polynomial densities*, *MFCS '92* (I. M. Havel and V. Koubek, editors), Lecture Notes in Computer Science, vol. 629, Springer, 1992, pp. 494–503.
- [54] J. W. THATCHER and J. B. WRIGHT, *Generalized finite automata theory with an application to a decision problem of second-order logic*, *Mathematical Systems Theory*, vol. 2 (1968), pp. 57–81.
- [55] W. THOMAS, *Automata on infinite objects*, *Handbook of theoretical computer science* (J. van Leeuwen, editor), vol. B: Formal models and semantics, Elsevier, 1990, pp. 133–191.
- [56] B. A. TRAHENBROT, *Finite automata and the logic of one-place predicates*. Russian, *Siberian Mathematical Journal*, vol. 3 (1962), pp. 103–131, English translation: *American Mathematical Society Translations, Series 2*, vol. 59 (1966), pp. 23–55.
- [57] R. VILLEMAIRE, *The theory of $\langle N, +, V_k, V_l \rangle$ is undecidable*, *Theoretical Computer Science*, vol. 106 (1992), pp. 337–349.

DEPARTMENT OF COMPUTER SCIENCE
 UNIVERSITY OF AUCKLAND, NEW ZEALAND
E-mail: rubin@cs.auckland.ac.nz

See discussions, stats, and author profiles for this publication at: <https://www.researchgate.net/publication/228594972>

Automata-based presentations of infinite structures

Article · January 2009

DOI: 10.1017/CBO9780511974960.002

CITATIONS

10

READS

35

3 authors, including:



Vince Bárány

Google Inc.

22 PUBLICATIONS 203 CITATIONS

[SEE PROFILE](#)



Erich Graedel

RWTH Aachen University

158 PUBLICATIONS 3,780 CITATIONS

[SEE PROFILE](#)

All content following this page was uploaded by [Erich Graedel](#) on 12 April 2017.

The user has requested enhancement of the downloaded file. All in-text references [underlined in blue](#) are added to the original document and are linked to publications on ResearchGate, letting you access and read them immediately.

Automata-based presentations of infinite structures

Vince Bárány¹ and Erich Grädel² and Sasha Rubin³

¹ Oxford University Computing Laboratory
Wolfson Building, Parks Road, Oxford OX1 3QD, United Kingdom
vbarany@logic.rwth-aachen.de

² Mathematical Foundations of Computer Science
RWTH Aachen, D-52056 Aachen, Germany
graedel@logic.rwth-aachen.de

³ Department of Mathematics and Applied Mathematics
University of Cape Town, Private Bag, Rondebosch 7701, South Africa
srubin@math.cornell.edu

1

Automata-based presentations of infinite structures

Vince Bárány¹ and Erich Grädel² and Sasha Rubin³

1.1 Finite presentations of infinite structures

The model theory of finite structures is intimately connected to various fields in computer science, including complexity theory, databases, and verification. In particular, there is a close relationship between complexity classes and the expressive power of logical languages, as witnessed by the fundamental theorems of descriptive complexity theory, such as Fagin's Theorem and the Immerman-Vardi Theorem (see [78, Chapter 3] for a survey).

However, for many applications, the strict limitation to finite structures has turned out to be too restrictive, and there have been considerable efforts to extend the relevant logical and algorithmic methodologies from finite structures to suitable classes of infinite ones. In particular this is the case for databases and verification where infinite structures are of crucial importance [130]. *Algorithmic model theory* aims to extend in a systematic fashion the approach and methods of finite model theory, and its interactions with computer science, from finite structures to finitely-presentable infinite ones.

There are many possibilities to present infinite structures in a finite manner. A classical approach in model theory concerns the class of *computable structures*; these are countable structures, on the domain of nat-

¹ Oxford University Computing Laboratory
Wolfson Building, Parks Road, Oxford OX1 3QD, United Kingdom
vbarany@logic.rwth-aachen.de

² Mathematical Foundations of Computer Science
RWTH Aachen, D-52056 Aachen, Germany
graedel@logic.rwth-aachen.de

³ Department of Mathematics and Applied Mathematics
University of Cape Town, Private Bag, Rondebosch 7701, South Africa
srubin@math.cornell.edu

ural numbers, say, with a finite collection of computable functions and relations. Such structures can be finitely presented by a collection of algorithms, and they have been intensively studied in model theory since the 1960s. However, from the point of view of algorithmic model theory the class of computable structures is problematic. Indeed, one of the central issues in algorithmic model theory is the effective evaluation of logical formulae, from a suitable logic such as first-order logic (FO), monadic second-order logic (MSO), or a fixed point logic like LFP or the modal μ -calculus. But on computable structures, only the quantifier-free formulae generally admit effective evaluation, and already the existential fragment of first-order logic is undecidable, for instance on the computable structure $(\mathbb{N}, +, \cdot)$.

This leads us to the central requirement that for a suitable logic L (depending on the intended application) the model-checking problem for the class \mathcal{C} of finitely presented structures should be algorithmically solvable. At the very least, this means that the L -theory of individual structures in \mathcal{C} should be decidable. But for most applications somewhat more is required:

Effective semantics: There should be an algorithm that, given a finite presentation of a structure $\mathfrak{A} \in \mathcal{C}$ and a formula $\psi(\bar{x}) \in L$, expands the given presentation to include the relation $\psi^{\mathfrak{A}}$ defined by ψ on \mathfrak{A} .

This also implies that the class \mathcal{C} should be closed under some basic operations (such as logical interpretations). Thus we should be careful to restrict the model of computation. Typically, this means using some model of *finite automata* or a very restricted form of rewriting.

In general, the finite means for presenting infinite structures may involve different approaches: logical interpretations; finite axiomatisations; rewriting of terms, trees, or graphs; equational specifications; the use of synchronous or asynchronous automata, etc. The various possibilities can be classified along the following lines:

Internal: a set of finite or infinite words or trees/terms is used to represent the domain of (an isomorphic copy of) the structure. Finite automata/rewriting-rules compute the domain and atomic relations (eg. prefix-recognisable graphs, automatic structures).

Algebraic: a structure is represented as the least solution of a finite set of recursive equations in an appropriately chosen algebra of finite and countable structures (eg. VR-equational structures).

Logical: structures are described by interpreting them, using a finite collection of formulae, in a fixed structure (eg. tree-interpretable structures). A different approach consists in (recursively) axiomatising the isomorphism class of the structure to be represented.

Transformational: structures are defined by sequences of prescribed transformations, such as graph-unraveling, or Muchnik's iterations, applied to certain fixed initial structures (which are already known to have a decidable theory). Transformations can also be transductions, logical interpretations, etc. [23]

The last two approaches overlap somewhat. Also, the algebraic approach can be viewed *generatively*: convert the equational system into an appropriate *deterministic grammar* generating the solution of the original equations [44]. The grammar is thus the finite presentation of the graph. One may also say that internal presentations and generating grammars provide descriptions of the *local structure* from which the whole arises, as opposed to descriptions based on *global symmetries* typical of algebraic specifications.

Prerequisites and notation

We assume rudimentary knowledge of finite automata on finite and infinite words and trees, their languages and their correspondence to monadic second-order logic (MSO) [133, 79]. Undefined notions from logic and algebra (congruence on structures, definability, isomorphism) can be found in any standard textbook. We mainly consider the following logics \mathcal{L} : first-order (FO), monadic second order (MSO), and weak monadic second-order (wMSO) which has the same syntax as MSO, but the intended interpretation of the set variables is that they range over *finite* subsets of the domain of the structure under consideration.

We mention the following to fix notation: infinite words are called ω -words and infinite trees are called ω -trees (to distinguish them from finite ones); relations computable by automata will be called *regular*; the domain of a *structure* \mathfrak{B} is usually written B and its relations are written $R^{\mathfrak{B}}$. An MSO-formula $\phi(X_1, \dots, X_j, x_1, \dots, x_k)$ interpreted in \mathfrak{B} defines the set $\phi^{\mathfrak{B}} := \{(B_1, \dots, B_j, b_1, \dots, b_k) \mid B_i \subset B, b_i \in B, \mathfrak{B} \models \phi(B_1, \dots, B_j, b_1, \dots, b_k)\}$. A wMSO-formula is similar except that the B_i range over finite subsets of B . The *full binary tree* \mathfrak{T}_2 is defined as the structure

$$(\{0, 1\}^*, \mathbf{suc}_0, \mathbf{suc}_1)$$

where the successor relation \mathbf{suc}_i consists of all pairs (x, xi) . Tree automata operate on Σ -labelled trees $T : \{0, 1\}^* \rightarrow \Sigma$. Such a tree is identified with the structure

$$(\{0, 1\}^*, \mathbf{suc}_0, \mathbf{suc}_1, \{T^{-1}(\sigma)\}_{\sigma \in \Sigma}).$$

Rabin proved the decidability of the MSO-theory of \mathfrak{T}_2 and the following fundamental correspondence between MSO and tree automata (see [132] for an overview):

For every monadic second-order formula $\varphi(\bar{X})$ in the signature of \mathfrak{T}_2 there is a tree automaton \mathcal{A} (and vice versa) such that

$$L(\mathcal{A}) = \{T_{\bar{X}} \mid \mathfrak{T}_2 \models \varphi(\bar{X})\} \quad (1.1)$$

where $T_{\bar{X}}$ denotes the tree with labels for each X_i .

Similar definitions and results hold for r -ary trees, in which case the domain is $[r]^*$ where $[r] := \{0, \dots, r - 1\}$, and finite trees.

In section 1.2.2 and elsewhere we do not distinguish between a term and its natural representation as a tree. Thus we may speak of infinite terms. We consider countable, vertex- and edge-labelled graphs possibly having distinguished vertices (called sources), and no parallel edges of the same label. A graph is *deterministic* if each of its vertices is the source of at most one edge of each edge label.

Interpretations

Interpretations allow one to define an isomorphic copy of one structure in another. Fix a logic \mathcal{L} . A d -dimensional \mathcal{L} -interpretation \mathcal{I} of structure $\mathfrak{B} = (B; (R_i^{\mathfrak{B}})_i)$ in structure \mathfrak{A} , denoted $\mathfrak{B} \leq_{\mathcal{L}}^{\mathcal{I}} \mathfrak{A}$, consists of the following \mathcal{L} -formulas in the signature of \mathfrak{A} ,

- a domain formula $\Delta(\bar{x})$,
- a relation formula $\Phi_{R_i}(\bar{x}_1, \dots, \bar{x}_{r_i})$ for each relation symbol R_i , and
- an equality formula $\epsilon(\bar{x}_1, \bar{x}_2)$,

where each $\Phi_{R_i}^{\mathfrak{A}}$ is a relation on $\Delta^{\mathfrak{A}}$, each of the tuples \bar{x}_i, \bar{x} contain the same number of variables, d , and $\epsilon^{\mathfrak{A}}$ is a congruence on the structure $(\Delta^{\mathfrak{A}}, (\Phi_{R_i}^{\mathfrak{A}})_i)$, so that \mathfrak{B} is isomorphic to

$$(\Delta^{\mathfrak{A}}, (\Phi_{R_i}^{\mathfrak{A}})_i) / \epsilon^{\mathfrak{A}}.$$

If \mathcal{L} is FO then the free \bar{x} are FO and we speak of a *FO interpretation*. If \mathcal{L} is MSO (wMSO) but the free variables are FO, then we speak of a (*weak*) *monadic second-order interpretation*.

We associate with \mathcal{I} a transformation of formulas $\psi \mapsto \psi^{\mathcal{I}}$. For illustration we define it in the first-order case: the variable x_i is replaced by the d -tuple \bar{y}_i , $(\psi \vee \phi)^{\mathcal{I}}$ by $\psi^{\mathcal{I}} \vee \phi^{\mathcal{I}}$, $(\neg\psi)^{\mathcal{I}}$ by $\neg\psi^{\mathcal{I}}$, $(\exists x_i \psi)^{\mathcal{I}}$ by $\exists \bar{y}_i \Delta(\bar{y}_i) \wedge \psi^{\mathcal{I}}$, and $(x_i = x_j)^{\mathcal{I}}$ is replaced by $\epsilon(\bar{y}_i, \bar{y}_j)$. Thus one can translate \mathcal{L} formulas from the signature of \mathfrak{B} into the signature of \mathfrak{A} .

Proposition 1.1.1 *If $\mathfrak{B} \leq_{\mathcal{L}}^{\mathcal{I}} \mathfrak{A}$, say the isomorphism is f , then for every formula $\psi(x_1, \dots, x_k)$ in the signature of \mathfrak{B} and all k -tuples \bar{b} of elements of \mathfrak{B} it holds that*

$$\mathfrak{B} \models \psi(b_1, \dots, b_k) \iff \mathfrak{A} \models \psi^{\mathcal{I}}(f(b_1), \dots, f(b_k))$$

In particular, if \mathfrak{A} has decidable \mathcal{L} -theory, then so does \mathfrak{B} .

Set interpretations

When \mathcal{L} is MSO (wMSO) and the free variables are MSO (wMSO) the interpretation is called a *(finite) set interpretation*. In this last case, we use the notation $\mathfrak{B} \leq_{\text{set}}^{\mathcal{I}} \mathfrak{A}$ or $\mathfrak{B} \leq_{\text{fset}}^{\mathcal{I}} \mathfrak{A}$. We will only consider (finite) set interpretations of dimension 1.

If finiteness of sets is MSO-definable in some structure \mathfrak{A} (as for linear orders or for finitely branching trees) then every structure \mathfrak{B} having a finite-set interpretation in \mathfrak{A} can also be set interpreted in \mathfrak{A} .

Example 1.1.2 An interpretation $(\mathbb{N}, +) \leq_{\text{fset}}^{\mathcal{I}} (\mathbb{N}, 0, \text{suc})$ based on the binary representation is given by $\mathcal{I} = (\varphi(X), \varphi_+(X, Y, Z), \varphi_=(X, Y))$ with $\varphi(X)$ always true, φ_+ the identity, and $\varphi_+(X, Y, Z)$ is

$$\exists C \forall n [(Zn \leftrightarrow Xn \oplus Yn \oplus Cn) \wedge (C(\text{suc}n) \leftrightarrow \mu(Xn, Yn, Cn)) \wedge \neg C]$$

where C stands for carry, \oplus is exclusive or, and $\mu(x_0, x_1, x_2)$ is the majority function, in this case definable as $\bigvee_{i \neq j} x_i \wedge x_j$.

To every (finite) subset interpretation \mathcal{I} we associate, as usual, a transformation of formulas $\psi \mapsto \psi^{\mathcal{I}}$, in this case mapping first-order formulas to (weak) monadic second-order formulas.

Proposition 1.1.3 *Let $\mathfrak{B} \leq_{(\text{f})\text{set}}^{\mathcal{I}} \mathfrak{A}$ be a (finite) subset interpretation with isomorphism f . Then to every first-order formula $\psi(x_1, \dots, x_k)$ in the signature of \mathfrak{B} one can effectively associate a (weak) monadic second-order formula $\psi^{\mathcal{I}}(X_1, \dots, X_k)$ in the signature of \mathfrak{A} such that for all k -tuples \bar{b} of elements of \mathfrak{B} it holds that*

$$\mathfrak{B} \models \psi(b_1, \dots, b_k) \iff \mathfrak{A} \models \psi^{\mathcal{I}}(f(b_1), \dots, f(b_k)) .$$

Consequently, if the (weak) monadic-second order theory of \mathfrak{A} is decidable then so is the first-order theory of \mathfrak{B} .

For more on subset interpretations we refer to [23].

1.2 A hierarchy of finitely presentable structures

This section provides an overview of some of the prominent classes of graphs and their various finite presentations.

These developments are the product of over two decades of research in diverse fields. We begin our exposition with the seminal work of Muller and Schupp on context-free graphs, we mention prefix-recognisable structures, survey hyperedge-replacement and vertex-replacement grammars and their corresponding algebraic frameworks leading up to equational graphs in algebras with asynchronous or synchronous product operation. These latter structures are better known in the literature by their automatic presentations, and constitute the topic of the rest of this survey.

As a unifying approach we discuss how graphs belonging to individual classes can be characterised as least fixed-point solutions of finite systems of equations in a corresponding algebra of graphs. We illustrate on examples how to go from graph grammars through equational presentations and interpretations to internal presentations and vice versa.

We briefly summarise key results on Caucal's pushdown hierarchy and more recent developments on simply-typed recursion schemes and collapsible pushdown automata.

Figure 1.1 provides a summary of some of the graph classes discussed in this section together with the boundaries of decidability for relevant logics. Rational graphs and automatic graphs featured on this diagram are described in detail in Section 1.3.

1.2.1 From context-free graphs to prefix-recognisable structures

Context-free graphs were introduced in the seminal papers [110, 111, 112] of Muller and Schupp. There are several equivalent definitions. The objects of study are countable directed edge-labelled, finitely branching graphs. An *end* is a maximal connected⁴ component of the induced subgraph obtained by removing, for some n , the n -neighbourhood of a fixed

⁴ connectedness is taken with respect to the underlying undirected graph.

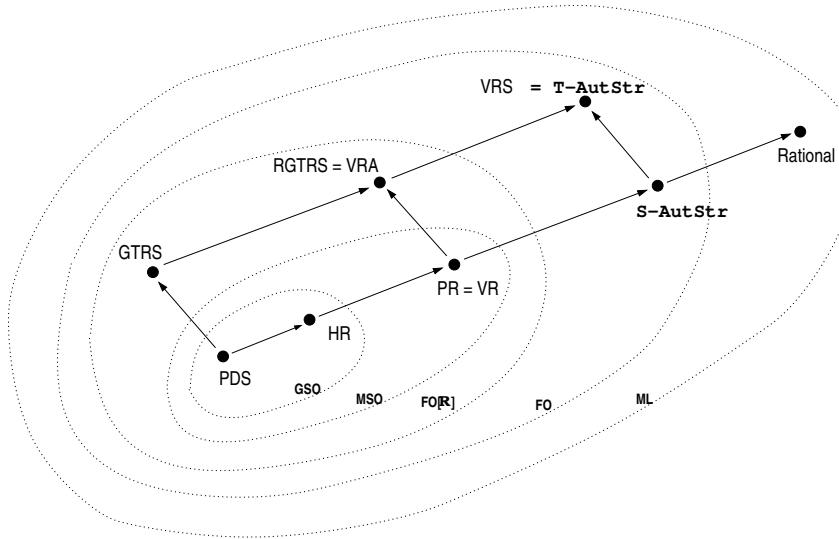


Figure 1.1 Relationship of graph classes and logical decidability boundaries.

vertex v_0 . A vertex of an end is on the *boundary* if it is connected to a vertex in the removed neighbourhood. Two ends are end-isomorphic if there is a graph isomorphism (preserving labels as well) between them that is also a bijection of their boundaries. A graph is *context-free* if it is connected and has only *finitely many ends* up to end-isomorphism. This notion is independent of the v_0 chosen.

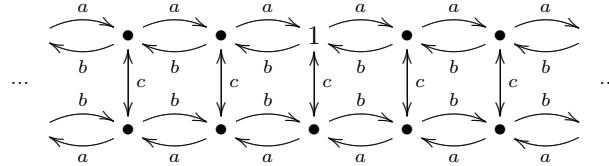
A graph is context-free if and only if it is isomorphic to the connected component of the configuration graph of a pushdown automaton (without ϵ -transitions) induced by the set of configurations that are reachable from the initial configuration [112].

A *context-free group* is a finitely generated group G such that, for some set S of semigroup generators of G , the set of words $w \in S^*$ representing the identity element of G forms a context-free language. This is independent of the choice of S . Moreover, a group is context-free if and only if its Cayley graph for some (and hence all) sets S of semigroup generators is a context-free graph. Finally, a finitely generated group is context-free if and only if it is *virtually free*, that is, if it has a free subgroup of finite index [111].⁵

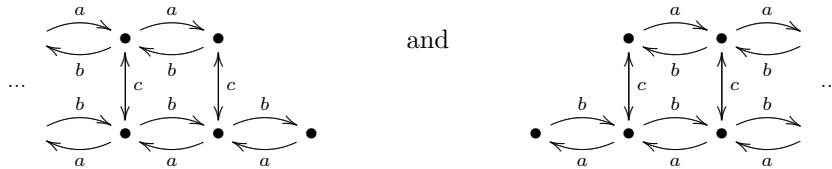
⁵ Originally [111] proved this under the assumption of *accessibility*, a notion related to group decompositions introduced by Wall who conjectured that all

Muller and Schupp have further shown that context-free graphs have a decidable MSO-theory. Indeed, every context-free graph can be MSO-interpreted in the full binary tree.

Example 1.2.1 Consider the group G given by the finite presentation $\langle a, b, c \mid ab, cc, acac, bcba \rangle$. The Cayley graph $\Gamma(G, S)$ of G with respect to the set of semigroup generators $S = \{a, b, c\}$ is depicted below.



Notice that $\Gamma(G, S)$ has two ends, for any n -neighbourhood of the identity with $n > 1$. These are



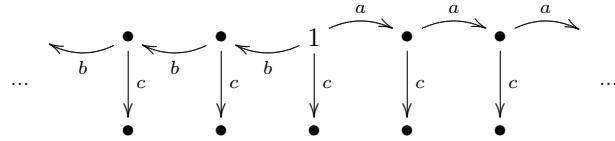
A word $w \in \{a, b, c\}^*$ represents the identity of G if, and only if, w has an even number of c 's and the number of a 's equals the number of b 's. We present a pushdown automaton \mathcal{A} which recognises this set of words and, moreover, has a configuration graph that is isomorphic to $\Gamma(G, S)$. The states of \mathcal{A} are $Q = \{1, c\}$ with $q_0 = 1$ as the initial state, the stack alphabet is $\Gamma = \{a, b\}$, the input alphabet is $\{a, b, c\}$ and \mathcal{A} has the following transitions:

$$\begin{array}{lll} \text{internal: } 1\theta & \xrightarrow{c} & c\theta \\ \text{internal: } c\theta & \xrightarrow{c} & 1\theta \\ \text{push: } q\sigma\theta & \xrightarrow{\sigma} & q\sigma\sigma\theta \quad \text{for } q = 1, c \text{ and } \sigma = a, b \\ \text{push: } q\perp & \xrightarrow{\sigma} & q\sigma\perp \quad \text{for } q = 1, c \text{ and } \sigma = a, b \\ \text{pop: } q\sigma\theta & \xrightarrow{\bar{\sigma}} & q\theta \quad \text{for } q = 1, c \text{ and } \{\sigma, \bar{\sigma}\} = \{a, b\} \end{array}$$

finitely generated groups would have this property. Muller and Schupp conjectured every context-free group to be accessible, but it was not until Dunwoody [64] proved that all finitely presentable groups are accessible that this auxiliary condition could be dropped from the characterisation of [111]. Unfortunately, many sources forget to note this fact. Later Dunwoody also gave a counterexample refuting Wall's conjecture.

Here θ is the stack content written with its top element on the left and always ending in the special symbol \perp marking the bottom of the stack.

In every deterministic edge-labelled connected graph and for any ordering of the edge labels one obtains a spanning tree by taking the shortest path with the lexicographically least labeling leading to each node from a fixed source. Take such a spanning tree T for the example graph $\Gamma(G, S)$ with root 1_G . Observe that T is regular, having only finitely many subtrees (ends) up to isomorphism. The ordering $a < b < c$ induces the spanning tree depicted below. The Cayley graph $\Gamma(G, S)$ is MSO-interpretable in this regular spanning tree by defining the missing edges using the relators from the presentation of the group.



In particular $\Gamma(G, S)$ is MSO-interpretable in the full binary tree, and hence has decidable MSO.

A mild generalisation of pushdown transitions, *prefix-rewriting* rules, take the form $uz \mapsto vz$ where u and v are fixed words and z is a variable ranging over words. As in the previous example, pushdown transitions are naturally perceived as prefix-rewriting rules affecting the state and the top stack symbols. Conversely, Caucal [40] has shown that connected components of configuration graphs of prefix-rewriting systems given by finitely many prefix-rewriting rules are effectively isomorphic to connected components of pushdown graphs. Later, Caucal introduced *prefix-recognisable graphs* as a generalisation of context-free graphs and showed that these are MSO-interpretable in the full binary tree and hence have a decidable MSO-theory [42].

Definition 1.2.2 (Prefix-recognisable relations) Let Σ be a finite alphabet. The set $\text{PR}(\Sigma)$ of prefix-recognisable relations over Σ^* is the smallest set of relations such that

- every regular language $L \subseteq \Sigma^*$ is a prefix-recognisable unary relation;
- if $R, S \in \text{PR}$ (arities r and s) and L is regular then $L \cdot (R \times S) = \{(uv_1, \dots, uv_r, uw_1, \dots, uw_s) \mid u \in L, \bar{v} \in R, \bar{w} \in S\} \in \text{PR}$;
- if $R \in \text{PR}$ of arity $m > 1$ and $\{i_1, \dots, i_m\} = \{1, \dots, m\}$,
then $R^{(\bar{i})} = \{(u_{i_1}, \dots, u_{i_m}) \mid (u_1, \dots, u_m) \in R\} \in \text{PR}$;
- if $R, S \in \text{PR}$ are of the same arity, then $R \cup S \in \text{PR}$.

Example 1.2.3 Consider the lexicographic ordering $<_{\text{lex}}$ on an ordered alphabet Σ . It is prefix-recognisable being the union of

$$\Sigma^* \cdot (\{\varepsilon\} \times \Sigma^+) \quad \text{and} \quad \Sigma^* \cdot (a\Sigma^* \times b\Sigma^*) \quad \text{for all } a < b \in \Sigma .$$

Following [22] we say that a structure $\mathfrak{A} = (A, \{R_i\}_i)$ is *prefix-recognisable* if A is a regular set of words over some finite alphabet Σ and each of the relations R_i is in $\text{PR}(\Sigma)$. Prefix-recognisable structures can be characterized in terms of interpretations. On the basis of tree automata, it is relatively straightforward to show that the prefix-recognisable structures coincide with the structures that are MSO-interpretable in the binary tree \mathfrak{T}_2 [97, 42, 22]. This result has been strengthened by Colcombet [51] to first-order interpretability in the expanded structure (\mathfrak{T}_2, \prec) (note that the prefix relation \prec is MSO-definable but not FO definable in \mathfrak{T}_2). Colcombet proved that MSO-interpretations and FO-interpretations in (\mathfrak{T}_2, \prec) have the same power, which gives a new characterisation of prefix-recognisable structures. We summarize these results as follows.

Theorem 1.2.4 *For every structure \mathfrak{A} , the following are equivalent.*

- (1) \mathfrak{A} is isomorphic to a prefix-recognisable structure;
- (2) \mathfrak{A} is MSO-interpretable in the full binary tree \mathfrak{T}_2 ;
- (3) \mathfrak{A} is FO-interpretable in (\mathfrak{T}_2, \prec) .

In particular, every prefix-recognisable structure has a decidable MSO-theory.

Below we discuss further characterisations of prefix-recognisable structures in terms of vertex-replacement grammars, or as least solutions of VR-equational systems.

1.2.2 Graph grammars and graph algebras

In this section we consider vertex- and edge-labelled graphs. In formal language theory grammars generate sets of finite words. Similarly, context-free graph grammars produce sets of finite graphs - start from an initial nonterminal and rewrite nonterminal vertices and edges according to the derivation rules. Just as for languages, the set of valid derivation trees, or parse trees, forms a regular set of trees labelled by derivation rules of the graph grammar. Conversely, consider a collection Θ of graph operations — such as disjoint union, recolourings, etc. — as primitives. Every closed Θ -term t evaluates to a finite graph $\llbracket t \rrbracket$, and similarly every

Θ -term $t(\bar{x})$ evaluates to a finite graph $\llbracket t(\bar{x}) \rrbracket$ with non-terminal (hyper)-edges and/or vertices. Formally, evaluation is the unique homomorphism from the initial algebra of Θ -terms to the Θ -algebra of finite graphs with non-terminals. Each regular tree language L of closed terms thus represents a family of finite graphs $\{\llbracket t \rrbracket \mid t \in L\}$. For a concise treatment of graph grammars and finite graphs we refer to the surveys [69, 59] and the book [53].

Our focus here is on individual countable graphs generated by *deterministic* grammars via ‘complete rewriting’. A suitable framework for formalising complete rewriting, in the context of term rewriting, is convergence in complete partial orders (cpo’s). Since no classical order- or metric-theoretic notion of limit seems to exist for graphs, we use the more general categorical notion of colimit [11]. We outline this framework in which an infinite term (over the graph operations Θ) yields a countable graph; details may be found in [55, 11, 53].

In the category \mathbb{G} of graphs and their homomorphisms every diagram of the form

$$G_0 \xrightarrow{f_0} G_1 \xrightarrow{f_1} G_2 \xrightarrow{f_2} \dots \xrightarrow{f_{n-1}} G_n \xrightarrow{f_n} G_{n+1} \xrightarrow{f_{n+1}} \dots$$

has a colimit G , i.e. a kind of least common extension G of the G_n s with homomorphisms $g_n : G_n \rightarrow G$ such that $g_n = g_{n+1}f_n$ for all n .⁶ We assume that the graph operations in Θ determine endofunctors of \mathbb{G} that are cocontinuous i.e. colimit preserving.

On the other side, take the cpo of finite and infinite terms over the signature $\Theta \cup \{\perp\}$, with the empty term \perp and the extension ordering $s \sqsubseteq t$. We may turn it into a category \mathbb{T}_Θ with each relation $s \sqsubseteq t$ inducing a unique arrow $s \rightarrow t$. Moreover, in this category, colimits (of diagrams as above) exist and an infinite term t is the colimit of approximations $t_0 \rightarrow t_1 \rightarrow \dots$ (think that t_i is the restriction of t to the first i levels). The evaluation mapping $\llbracket \cdot \rrbracket$ has a unique cocontinuous extension, also denoted $\llbracket \cdot \rrbracket$, mapping infinite terms to colimits of graphs.

This completes the basic description. Now consider a grammar \mathcal{G} whose derivation rules $\langle X_i \mapsto t_i(\bar{X}) \rangle$ can be expressed by Θ -terms. These terms determine cocontinuous endofunctors in the category of terms \mathbb{T}_Θ . By the Knaster-Tarski theorem the functors have a least fixed-point \overline{G} , which by Kleene’s Theorem is attained as the colimit of the chain

⁶ There are examples of ascending chains $G_0 \xrightarrow{f_0} G_1 \xrightarrow{f_1} \dots$ and $G_0 \xrightarrow{g_0} G_1 \xrightarrow{g_1} \dots$ with identical graphs but different embeddings yielding different colimits, whence there is no apparent canonical way of defining a limit knowing only that each G_n is embeddable into G_{n+1} .

$\langle \gamma^n(\bar{\emptyset}) \rangle_n$ with the natural homomorphisms. The graph *generated* by the grammar from the corresponding non-terminal X_i is defined to be the component G_i of the colimit \bar{G} .

Equivalently, given the system of equations $\mathcal{E}_G = \langle X_i = t_i(\bar{X}) \rangle$ one can construct a syntactic (uninterpreted) solution of \mathcal{E}_G by ‘unraveling’ these equations from the initial non-terminal X_0 of the grammar. This results in a possibly infinite regular term t_G , which is precisely the least fixed-point solution for X_0 in \mathbb{T}_Θ . By cocontinuity of the evaluation mapping $\llbracket t_G \rrbracket$ is isomorphic to the least fixed-point solution of \mathcal{E}_G in \mathbb{G} , that is to the graph generated by \mathcal{G} .

In what follows we focus on different sets of graph operations Θ (namely, HR, VR and some extensions). It has been observed that for suitable choices of operations, most notably avoiding products, the evaluation mapping can be realised as a monadic second-order interpretation or transduction [11, 60]. Consequently every interpretation $\llbracket t \rrbracket \leq_{\text{MSO}}^{\mathcal{I}} t$ naturally translates to an internal presentation of $\llbracket t \rrbracket$ using tree automata. Moreover, for a regular term t the MSO-theory of $\llbracket t \rrbracket$ is decidable by Rabin’s Theorem.

Finally we mention that all this smoothly extends to solutions of infinite sets of equations [33]. Although unravelling might not result in a regular solution term, as long as it has a decidable MSO-theory so does the solution graph.

Equational graphs and hyperedge-replacement grammars

Hyperedge-replacement (HR) grammars are a very natural generalisation of context-free grammars from formal language theory. Every HR-grammar defines a ‘language’ of finite graphs just as context-free grammars define languages of finite words. The class of graph languages defined by HR-grammars possesses many structural properties akin to those well-known for context-free languages. The interested reader is referred to the monograph [80].

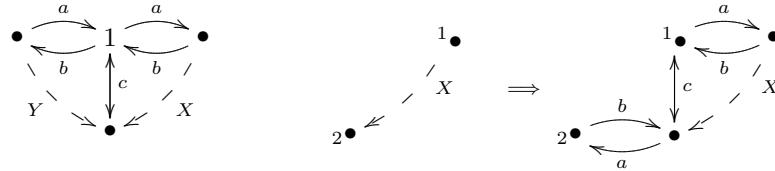
An HR-grammar is given as a finite collection of rules that allow the replacement of any hyperedge of a hypergraph bearing a non-terminal label by the right hand side of a matching rule, which is a given finite hypergraph with a number of distinguished vertices equal to the arity of the hyperedge to be replaced. A copy of the right-hand side of a matching rule is then glued to the original hypergraph precisely at these distinguished vertices and corresponding end vertices of the hyperedge being replaced. Derivation begins with a distinguished non-terminal.

As outlined at the start of section 1.2.2, each *deterministic* HR-grammar

determines a unique countable graph constructed from the initial graph by complete rewriting in the course of which every non-terminal hyperedge is eventually replaced by the right-hand side of the unique matching rule. A countable graph is **HR-equational**, or simply *equational*, if it is generated by a deterministic HR grammar [55]. The class of equational graphs will be denoted by **HR**. Equational graphs constitute a proper extension of the class of context-free graphs [41].

Proposition 1.2.5 *A connected graph is context-free if, and only if, it is equational and of finite degree.*

Example 1.2.6 To generate the context-free graph of Example 1.2.1 with a deterministic HR grammar we take as our initial graph the 1-neighbourhood of the root node (labelled with 1 above) and attach to it non-terminal hyperedges labelled with X and with Y , respectively, whose vertices enumerate the boundaries of either ends. Similarly, the 1-neighbourhood of the boundary of each end, that is the vertices of the corresponding non-terminal hyperedge, constitutes the right-hand side of the matching rule. Again, non-terminal hyperedges are attached to mark the new boundary. The initial graph and the rule for the non-terminal X obtained this way are pictured below.



Notice how the linearity of the generated graph is reflected in the linearity of the replacement rules each having only a single non-terminal hyperedge on the right. In the next example a non-linear rule is used to generate a tree, which is not context-free.

Example 1.2.7 The complete bipartite graph $K_{1,\omega}$ and the full ω -branching tree \mathfrak{T}_ω (in the signature of graphs) are not context-free, but can be generated by the following rules from the initial graph $\bullet \dashrightarrow \bullet$.



The HR-algebra of finite and countable graphs corresponding to hyperedge-replacement grammars is a many-sorted algebra defined as follows.

For each n there is a separate sort \mathbb{G}_n of graphs with n sources. These are distinguished vertices, though not necessarily distinct, named v_1, \dots, v_n . There are constants of each sort \mathbb{G}_n : these are hypergraphs having at most one hyperedge, exactly n vertices, each vertex a distinct source. The HR-algebra is built on the following operations: *disjoint union* \oplus , *renaming of sources* $\text{rename}_{c \rightarrow c'}$, and *fusion of sources* fuse_{\approx} according to an equivalence \approx on source names. By convention \oplus is understood to automatically shift the source names of its second argument by the maximum of the source names used in the first to avoid naming conflict. Also fuse assigns the least source name of a class to each fused node while dropping the others.

It is intuitively clear how a hyperedge-replacement step can be expressed using disjoint union with the right-hand side of the rule followed by a fusion and renaming of sources. Formally, one transforms an HR-grammar \mathcal{G} into a system of finitely many equations $X_i = t_i(\bar{X})$ where variables play the role of non-terminals of the grammar and the terms t_i are chosen such that, when variables are interpreted as individual hyperedges, $\llbracket t_i(\bar{X}) \rrbracket$ is the right hand side of the matching rule for a hyperedge labelled X_i .

Example 1.2.8 The equation corresponding to the single rule of the HR grammar of Example 1.2.7 generating \mathfrak{T}_ω is

$$X = \text{rename}_{0 \rightarrow 0, 1 \mapsto 1}(\text{fuse}_{\{0,2\}, \{1,4\}}(\overset{0}{\bullet} \rightarrow \overset{1}{\bullet} \oplus X \oplus X)).$$

Note that the source names of the first and second occurrences of X are shifted by 2 and by 4, respectively, while forming their disjoint union. Thus, after fusion we obtain precisely the right hand side of the HR-rule generating \mathfrak{T}_ω , however, with additional source names. The renaming operation in this term has the effect of forgetting the source names 2 and above. So the least solution of this equation is indeed \mathfrak{T}_ω with its root labelled 0 and one of its children with 1.

The generating power of HR-grammars is limited by the fact that edges can only be ‘created’ via fusion of sources (after having taken the disjoint union of two graphs). Because there are only a fixed number of source names available in a finite HR-equational system there is a bound on the size of complete bipartite subgraphs $K_{n,n}$ that can be created [12], cf. Theorem 1.2.12. The infinite bipartite graph $K_{\omega,\omega}$ is thus an example of a prefix-recognisable graph which is not HR-equational.

It is a key observation that in case of HR-terms the evaluation mapping

$t \mapsto \llbracket t \rrbracket$ is expressible as an MSO-interpretation. In fact, since edges cannot be created by any of the HR operations, the vertex-edge-adjacency graph of $\llbracket t \rrbracket$ is MSO-interpretable in the tree representation of t , whether t is finite or infinite.

Theorem 1.2.9 *For a countable graph G the following are equivalent.*

- (1) G is generated by a deterministic HR grammar;
- (2) G is HR-equational, i.e. the evaluation of a regular HR-term, i.e. the least solution of a finite system of HR-equations;
- (3) The two-sorted incidence graph \hat{G} of G is monadic second-order interpretable in the full binary tree, i.e. $\hat{G} \leq_{\text{MSO}} \mathfrak{T}_2$.

For a detailed presentation of these and other algebraic frameworks and their connections to the generative approach based on graph grammars we advise consulting [55, 12, 21]. In [54] Courcelle considered an extension of monadic second-order logic, denoted CMSO₂, in which one can quantify over sets of edges as well as over sets of vertices and, additionally, make use of modulo counting quantifiers. Notice that the last item of the previous theorem implies that the CMSO₂-theory of equational graphs is interpretable in S2S and is thus decidable. Further, Courcelle proved that CMSO₂ is able to axiomatise each and every equational graph up to isomorphism.

Theorem 1.2.10 *Each HR-equational graph is axiomatisable in CMSO₂. Consequently the isomorphism problem of equational graphs is decidable.*

Sénizergues considered HR-equational graphs of finite out-degree and proved that they are, up to isomorphism, identical with the ε -closures of configuration graphs of normalised⁷ pushdown automata restricted to the set of reachable configurations. Further, he proved that bisimulation equivalence of HR-equational graphs of finite out-degree is decidable [128]. This last result is an improvement on the decidability of bisimulation equivalence for deterministic context-free processes, which is a consequence of the celebrated result of Sénizergues establishing decidability of the DPDA language equivalence problem.

⁷ Here a PDA is said to be normalised, if in addition to being in a familiar normal-form its ε -transitions may not push anything on the stack. Hence the finiteness bound on the out-degree of configurations. For precise definitions see [128].

Vertex-replacement grammars

Vertex replacement systems are a finite collection of graph rewriting rules that allow one to substitute given finite graphs in place of single vertices while keeping all the connections. This form of graph rewriting emerged as the most robust and manageable from among a host of different notions within a very general framework [55, 69, 59, 58]. The corresponding VR-algebra of graphs is built on the following operations: constant graphs of a single c -coloured vertex \bullet^a , disjoint union \oplus , recolouring of vertices $\text{recol}_{c \leftrightarrow c'}$ and introduction of a -coloured edges $\text{edge}_{c \xrightarrow{a} d}$ from every c -coloured vertex to every d -coloured vertex.

The evaluation of VR-terms, whether finite or infinite, is realisable as a monadic second-order interpretation. More precisely, as VR-equational graphs are interpretations of regular terms obtained by unfolding a finite system of VR equations, they can be MSO-interpreted in a regular tree, hence also in the full binary tree \mathfrak{T}_2 , and thus are prefix-recognisable. These and other characterisations, together with our previous discussion of prefix-recognisable structures are summarised in the next theorem.

Theorem 1.2.11 *For a countable graph G the following are equivalent.*

- (1) G is isomorphic to a prefix-recognisable structure;
- (2) G is generated by a deterministic VR grammar;
- (3) G is VR-equational, i.e. the evaluation of a regular VR-term, i.e. the least solution of a finite system of equations of the form $X_i = t_i(\overline{X})$ with finite VR-terms $t_i(\overline{X})$;
- (4) $G \leq_{\text{MSO}} \mathfrak{T}_2$;
- (5) $G = h^{-1}(\mathfrak{T}_2)|_C$, i.e. the vertices of G are obtained by restricting the nodes of \mathfrak{T}_2 to a regular set C , and its edges are obtained by taking the inverse of a rational substitution h to \mathfrak{T}_2 ;
- (6) G is isomorphic to the ϵ -closure of the configuration graph of a push-down automaton.

Further, the HR-equational graphs can be characterised as the class of VR-equational graphs of finite tree width [11].

Theorem 1.2.12 *VR-equational graphs of finite tree width are HR-equational.*

Example 1.2.13 The complete bipartite graph $K_{\omega,\omega}$ is a prominent example of a VR-equational graph that is not HR-equational. A VR grammar and the corresponding system of VR equations generating $K_{\omega,\omega}$ are

given below.

$$\begin{array}{ccc} \overset{X}{\bullet} & \Rightarrow & \overset{A}{\bullet} \xleftarrow{\quad} \overset{A}{\bullet} \\ \overset{A}{\bullet} & \Rightarrow & \bullet \overset{A}{\bullet} \end{array} \quad \begin{array}{lcl} X & = & \text{edge}_{a \leftrightarrow b}(A \oplus \text{recol}_{a \mapsto b}(A)) \\ A & = & \overset{a}{\bullet} \oplus A \end{array}$$

The expressive power of this formalism (for describing families of finite graphs) is not increased by extending the VR operations by graph transformations that are definable using quantifier-free formulas (of which $\text{recol}_{c \rightarrow c'}$ and $\text{edge}_{c \xrightarrow{a} d}$ are particular examples), nor by the *fusion* operations fuse_c identifying all nodes bearing a certain colour c [60]. Care has to be taken when defining countable graphs as evaluations of infinite terms, for it is unclear how to deal with infinite terms built with non-monotonic operations. Nonetheless, infinite terms built with operations definable by *positive* quantifier-free formulas can be evaluated unambiguously [11].

In this setting Theorem 1.2.11 can be generalised to infinite systems of equations (whose unfoldings are typically non-regular terms) using infinite deterministic automata [33], leading us to the following families of transition graphs.

1.2.3 Higher-order data structures

Tree-constructible graphs and Caucal's pushdown hierarchy

Courcelle introduced MSO-*compatible transductions* in the investigation of structures with decidable monadic theories. Let \mathcal{C} and \mathcal{C}' be classes of structures on signatures σ and σ' , respectively. Following [57] we say that a functional transduction $T : \mathcal{C} \rightarrow \mathcal{C}'$ is MSO-compatible if there is an algorithm mapping each monadic formula φ of signature σ' to a monadic formula φ^T in the signature σ such that

$$\mathfrak{A} \models \varphi^T \iff T(\mathfrak{A}) \models \varphi.$$

MSO-interpretations are the most natural examples of MSO-compatible transductions. Slightly more generally, the MSO-*definable transductions* of Courcelle are MSO-compatible. Recall that these are given by a k -copying operation (for some k) followed by an MSO-interpretation and in particular the resulting structure may have k times the cardinality of the original one.

The more difficult result that the *unfolding* operation, mapping graphs (\mathfrak{G}, v) to trees $\mathfrak{T}_{(\mathfrak{G}, v)}$, is also MSO-compatible appeared in [61] (see also [57] for an exposition and a treatment of the simpler case of deterministic

graphs). We note that this result also follows from Muchnik's Theorem [126, 138, 17] and that it generalises Rabin's theorem.

A rich class of graphs, each with decidable monadic theory, can now be constructed. Caucal [43] proposed the hierarchies of graphs and trees obtained by alternately applying unfoldings and MSO-interpretations starting with finite graphs:

Definition 1.2.14

$$\begin{aligned}\text{Graphs}_0 &= \{\text{finite edge- and vertex-labelled graphs}\} \\ \text{Trees}_{n+1} &= \{\mathcal{T}_{\mathfrak{G}, v} \mid (\mathfrak{G}, v) \in \text{Graphs}_n\} \\ \text{Graphs}_{n+1} &= \{\mathcal{I}(\mathcal{T}) \mid \mathcal{T} \in \text{Trees}_{n+1}, \mathcal{I} \text{ is an MSO interpretation}\}\end{aligned}$$

By the results above, we have

Theorem 1.2.15 *For every $n \in \mathbb{N}$ every graph G from Graphs_n has a decidable MSO-theory.*

Fratani [72, 73] provided an alternative proof of the above theorem, among a host of other results on higher-order pushdown graphs, using a different kind of MSO-compatible operation. Indeed, she established that if a homomorphism of words maps the branches of a tree T to those of T' surjectively while also preserving the node-labeling then definability and decidability results for MSO over T' can be transferred to T .

The Caucal hierarchy is very robust. Various weakenings and strengthenings of the definition yield exactly the same classes [37]. In fact, in place of MSO-interpretations, Caucal originally used inverse rational mappings in the style of item (5) of Theorem 1.2.11. Recently Colcombet [51] proved that every graph of Graphs_{n+1} can in fact be obtained via a first-order interpretation in some tree belonging to Trees_{n+1} . The next theorem provides internal presentations of graphs of each level as a generalisation of Theorem 1.2.11 item (6) thereby justifying the name pushdown hierarchy.

Theorem 1.2.16 ([37]) *For every n a graph G is in Graphs_n if, and only if, it is isomorphic to the ϵ -closure of the configuration graph of a higher-order pushdown automaton at level n .*

The strictness of the hierarchy was also shown in [37]. The level-zero graphs are the finite graphs, trees at level one are the regular trees, and as we have seen in Theorem 1.2.11 the level-one graphs are the prefix-recognisable ones. The deterministic level-two trees are known as

algebraic trees. From the second level onwards we have no clear structural understanding of the kind of graphs that inhabit the individual levels. We recommend [134] for an exposition.

Term-trees defined by recursion schemes

Caucal also gave a kind of algebraic characterisation of term-trees at level n as fixed points of *safe* higher-order recursion schemes.

Theorem 1.2.17 ([43]) *For every n , the class of term-trees Trees_n coincides with that of term-trees generated by safe higher-order recursion schemes of level at most n .*

The notion of higher-order schemes is a classical one [62, 56]. Safety is a technical restriction (implicit in [62]) ensuring that no renaming of variables (α -conversion) is needed during the generative substitutive reduction (β -reduction) process constructing the solution-term [1, 117]. Safe schemes are intimately related to the pushdown hierarchy. This connection is well explained in [1] showing that while on the one hand order- n schemes can define the behaviour and hence (the unfolding of) the configuration graphs of level- n deterministic pushdown automata, on the other hand, deterministic pushdown automata of level n can evaluate safe order- n schemes. Safety is hereto essential.

In order to evaluate arbitrary schemes [81] introduced *higher-order collapsible pushdown automata* (CPDA), a kind of generalisation of panic automata [92], and gave in essence the following characterisation in the spirit of Theorem 1.2.16.

Theorem 1.2.18 *The term-trees defined by order- n recursion schemes are up to isomorphism identical with the unfoldings of ϵ -closures of configuration graphs of level- n collapsible higher-order pushdown automata.*

As shown in [117, 81], it is not necessary to assume safety for establishing decidability of the MSO-theories of term-trees that are solutions of higher-order schemes.

Theorem 1.2.19 *The MSO-theory of a term-tree defined by an arbitrary higher-order recursion scheme is decidable.*

Consequently, configuration graphs of higher-order collapsible pushdown automata can be model-checked against modal μ -calculus formulas. However, there is a second-order CPDA whose configuration graph interprets the infinite grid and whose MSO-theory is thus undecidable

[81]. This shows that higher-order CPDA configuration graphs constitute a proper extension of Caucal's pushdown hierarchy.

1.2.4 Introducing products

There is a connection between the internal presentations of graphs seen so far and the graph operations used in the corresponding equational framework. Pushdown stacks are naturally represented as strings. The set of strings over some alphabet can in turn be modelled as an algebra of terms built with unary functions, one for each letter of the alphabet. Strings thus correspond to terms and letters to unary functions. In functional programming terminology the abstract data type of, say, binary strings has the recursive type definition

$$\mathcal{T} = \perp \oplus 0(\mathcal{T}) \oplus 1(\mathcal{T}) \quad (1.2)$$

Here the letters 0 and 1 are seen as type constructors and the empty string \perp is a constant type constructor. The set of finite strings is the least fixed-point solution of this equation.

Automata operating on terms of type \mathcal{T} can be viewed as functions mapping terms to states. Moreover these functions are defined according to structural recursion. Analogously, recursion schemes (fix-point equations) in an algebra of graph operations transform automata-based internal presentations of a graph into equational specifications. We can use the recursion scheme associated to the type definition (1.2) to define any PR-graph by a VR equation extending the type definition. For instance, the graph of the lexicographic order from Example 1.2.3 satisfies the following equation

$$L = \text{edge}_{0 \rightarrow 1, \varepsilon \rightarrow 0, \varepsilon \rightarrow 1}(\bullet^\varepsilon \oplus \text{recol}_{0,1,\varepsilon \mapsto 0}(L) \oplus \text{recol}_{0,1,\varepsilon \mapsto 1}(L)).$$

We briefly explain how to go from automata presenting a PR-graph to a VR-equation. For a language $V \subset \{0,1\}^*$ recognised by an automaton with transition table $\Delta \subset Q \times \Sigma \times Q$ and final states F the following VR-equation colours each word $w \in \{0,1\}^*$ by those states q such that the automaton starting from q accepts w . (N.B. in accordance with (1.2) the simulation proceeds right-to-left.)

$$X = \bullet^F \oplus \text{recol}_{\{q' \mapsto q : \Delta(q, 0, q')\}}(X) \oplus \text{recol}_{\{q' \mapsto q : \Delta(q, 1, q')\}}(X)$$

In general, every PR-graph $\bigcup_i U_i \cdot (V_i \times W_i)$ is the recolouring of a graph satisfying a VR-equation of the form

$$X = \vartheta(\vartheta_\varepsilon(\bullet) \oplus \vartheta_0(X) \oplus \vartheta_1(X)) . \quad (1.3)$$

Here, the states of the automata recognising V_i or W_i are encoded as vertex colours (just as above) and ϑ_ε colours \bullet by the final states of the V_i 's and W_i 's. Edge colours are used to represent states of automata for each U_i . For every $v \in V_i$ and $w \in W_i$, and z accepted by the automaton for U_i from state q there is a q -coloured edge (zv, zw) . To this end, ϑ_0 and ϑ_1 recolour the vertices and edges, and ϑ adds an edge between all $x \in V_i$ and $y \in W_i$ coloured by the final states of U_i .

In passing we mention that higher-order stacks can also be represented as strings: either as well-bracketed sequences of stack symbols, or as strings of stack operations yielding the particular stack configuration. The former comes at the cost of losing regularity of the domain and has no apparent algebraic counterpart. The latter gives rise to a unary algebra of higher-order stacks that is not, except for level 1 pushdown stacks, freely generated by the stack operations. Thus there is no unique term representing a general stack. The work of Fratani, Carayol and others [72, 73, 33, 32] has shown that both of these deficiencies can be turned into features.

We now turn to graphs internally presented by finite trees. A type definition for $\{0, 1\}$ -labelled binary branching trees is

$$\mathcal{T} = \perp \oplus 0(\mathcal{T} \otimes \mathcal{T}) \oplus 1(\mathcal{T} \otimes \mathcal{T}) \quad (1.4)$$

where \otimes denotes direct product. Later we will compare this with another type definition (1.6). Colcombet observed that this schema can be used to define graphs with internal presentations involving tree automata operating on finite trees. He proposed extensions of the VR-algebraic framework by the *asynchronous product* \otimes_A [48] and by the *synchronous product* \otimes_S [50, 49] which we shall denote here by VRA and VRS, respectively.

Definition 1.2.20 (Synchronous and asynchronous product) The products are defined for vertex and edge-coloured graphs \mathcal{G} and \mathcal{H} as follows. In the synchronous product there is a d -coloured edge from (g, h) to (g', h') if, and only if, both (g, g') and (h, h') are connected by a d -edge in \mathcal{G} and \mathcal{H} , respectively. The edge relation E_d of the asynchronous product $\mathcal{G} \otimes_A \mathcal{H}$ is defined as the union of $\{((g, h), (g', h')) \mid E_d^{\mathcal{G}}(g, g'), h \in H\}$ and $\{((g, h), (g, h')) \mid E_d^{\mathcal{H}}(h, h'), g \in G\}$. The definition of vertex colours requires a little care. In both cases a vertex (g, h) of the product has colour $\delta(c, c')$ whenever g has colour c and h has colour c' . Here the function $\delta : C^2 \rightarrow C$ is a parameter of the product operation. However,

it is really only relevant that δ acts as a pairing function on some sufficiently large subsets of the colours. For instance, Colcombet identifies C with $\{0, 1, \dots, N - 1\}$ and defines δ as addition modulo N [48].

As before, VRA-equational and VRS-equational graphs are defined as least fixed-point solutions of a finite system of equations in the respective algebra. Both product operations are cocontinuous with respect to graph embeddings. Therefore the evaluation mapping of both VRA and VRS terms uniquely extends from finite terms to infinite terms. Hence, just as for HR- and VR-equational graphs, the solution of a system of VRA or VRS equations is the evaluation of the regular term obtained by unraveling the system of equations.

Example 1.2.21 The infinite two-dimensional grid $(\mathbb{N} \times \mathbb{N}, \text{Up}, \text{Right})$ is easily constructed as the asynchronous product of the VR-equational, even context-free, graphs (\mathbb{N}, Up) and $(\mathbb{N}, \text{Right})$:

$$\begin{aligned} G &= \otimes_A(N_u, N_r) \\ N_u &= \text{edge}_{a \xrightarrow{\text{up}} b}^a \left(\bullet \oplus \text{recol}_{a \mapsto b, b \mapsto c}(N_u) \right) \\ N_r &= \text{edge}_{a \xrightarrow{\text{right}} b}^a \left(\bullet \oplus \text{recol}_{a \mapsto b, b \mapsto c}(N_r) \right) \end{aligned}$$

The unfolding of this system of equations is, schematically, an infinite term consisting of two periodic branches joined at the root. Elements of the grid G , by definition of asynchronous product, are represented as pairs of nodes of this term-tree with one node on either branch, corresponding to the respective co-ordinates. The example of the grid, whose MSO theory is undecidable, shows that the evaluation mapping of VRA terms (also of VRS terms) can not be realised by an MSO-interpretation.

For any VRA or VRS-term t , vertices of $\llbracket t \rrbracket$ can be identified with maximal *subsets* of nodes of t belonging to sub-terms joined by a product operator. It is thus easily expressible in MSO whether a set X of nodes (finite or infinite⁸) is actually well-formed in this sense, i.e. whether it represents an element of $\llbracket t \rrbracket$.

VR with asynchronous product and ground term rewriting

Ground term rewrite systems (GTRSs) are a natural generalisation of prefix-rewriting to trees. They are term rewrite systems given by rewrit-

⁸ In least fixed-point semantics only finite sets are considered, whereas in greatest fixed-point semantics both finite and infinite sets can represent elements of the solution, provided that there is an infinite nesting of product operators in t .

ing rules in which no variables occur. Tree automata are a special case of GTRSs (see [52]).

Example 1.2.22 The rewrite rule $a \rightarrow f(a)$ confined to terms of the form $d(f^n(a), f^m(a))$ is a GTRS whose configuration graph is isomorphic to the infinite square grid.

We have noted that prefix-recognisable graphs are identical to ε -closures of pushdown graphs. This correspondence is achieved by generalising the simple prefix-rewriting rules of pushdown systems of the form $v \rightarrow w$ where v and w are strings to replacement rules $V \rightarrow W$ for given regular languages V, W . The latter rule allows one to rewrite any prefix $v \in V$ of a given string by any word from W . Regular Ground Term Rewrite Systems (RGTRS) generalise GTRS in the exact same manner: simple ground rewrite rules $s \rightarrow t$ with ground terms s, t are replaced by ‘rule schemes’ $S \rightarrow T$ with regular sets of terms on both left and right-hand side.

Löding [99, 100] and Colcombet [48] studied transition graphs of GTRSs and RGTRSs from a model-checking point of view. In Löding’s work vertices of the transition graph are those terms reachable from an initial term, whereas Colcombet considers all terms of a given type as vertices.

The VR-equations defining PR graphs (1.3) easily generalise to VRA-equations defining graphs of RGTRSs using the recursion scheme (1.4):

$$X = \vartheta(\vartheta_\varepsilon(\bullet) \oplus \vartheta_0(X \otimes_A X) \oplus \vartheta_1(X \otimes_A X)) \quad (1.5)$$

For each rule $S_i \rightarrow T_i$ of the RGTRS we simulate (frontier to root) tree automata recognising S_i and T_i . Vertices of X represent terms, so we call these vertex-terms. A vertex-term is coloured by those states q occurring at the root of the term after being processed by the automata. The simulation is initialised as follows: ϑ_ε labels \bullet by initial states, and ϑ adds edges between all vertex-terms coloured by accepting states of automata for S_i and T_i . Updates occur in ϑ_j s according to the transition rules, similarly to (1.3). To this end assume that two vertex-terms v', v'' are coloured by states q' and q'' respectively. After taking the product the paired vertex-term $j(v', v'')$ is initialised with colour (q', q'') (cf. Def. 1.2.20). This pair is then recoloured to q by ϑ_j whenever (q, j, q', q'') is a transition.

Notice how naturally the asynchronous product captures closure of RGTR rewriting under contexts: if there was an edge between v and v' then there is an edge between $j(v, v'')$ and $j(v', v'')$, and, symmetrically,

between $j(v'', v)$ and $j(v'', v')$. One obtains along these lines the following generalisations of Theorem 1.2.11 (cf. examples 1.2.22 and 1.2.21).

Theorem 1.2.23 (Colcombet [48])

- (i) *A countable graph is VRA-equational if, and only if, it is (after removal of certain colours) isomorphic to an RGTRS graph⁹.*
- (ii) *Each VRA-equational graph is finite-subset interpretable in a regular term-tree, hence also in the full binary tree.*

Theorem 1.2.12 also extends to VRA-equational graphs [48, 100].

Theorem 1.2.24 *VRA-equational graphs of finite tree-width are HR-equational.*

An immediate consequence of Theorem 1.2.23 is that the FO-theory of every VRA-equational structure is decidable via interpretation in S2S. In fact, for any VRA-equational graph $G = (V, \{E_a\}_a)$ the subset interpretation, hence also first-order decidability, extends to G with additional reachability predicates $R_C = \{(v, w) \mid w \text{ can be reached from } v \text{ using edges of colours from } C\}$ for arbitrary subsets C of edge colours [48].

Theorem 1.2.25 *VRA-equational graphs have a decidable first-order theory with reachability.*

This result cannot be improved much further. Examples of [139] show that ‘regular reachability’, i.e. the problem whether there exists a path in a given VRA-equational graph between two given nodes and such that the labeling of the path belongs to a given regular language over the set of colours, is undecidable. In [100] Löding identified a maximal fragment of CTL that is decidable on every GTRS graph (with vertices restricted to terms reachable from an initial one) that can express, besides reachability, recurring reachability.

VR with synchronous product and tree-automatic structures

We have remarked that in the subset interpretation of VRA terms the subsets are used in a special form. Indeed, in the evaluating interpretation they merely serve the purpose of outlining the shape of a finite term. General finite-subset interpretations are more powerful and are capable of expressing the evaluation of VRS terms. In fact, these two formalism are equally expressive.

⁹ Here RGTRS graphs are taken in the sense of [48] as being restricted to the set of terms of a given type.

This is best explained by *tree-automatic presentations*. These are internal presentations of VRS-structures which will be formally introduced in the next section. For now it suffices to use the characterisation (Theorem 1.3.18) that tree-automatic graphs are those that are wMSO-interpretable in a regular tree (reflected in the equivalence of (1) and (2) below).

Theorem 1.2.26 (Colcombet [50])

For every countable graph G the following are equivalent

- (1) G is isomorphic to a tree-automatic graph.
- (2) G is interpretable in a regular tree (wlog. the full binary tree) via a finite-subset interpretation.
- (3) G is the restriction of a VRS-equational vertex-labelled graph G' to its set of vertices of a given colour;

We have noted that the evaluation mapping of VRS-terms can be naturally defined as a finite subset interpretation - this justifies (3) \rightarrow (2). Continuing our discussion of translations from automata-based internal presentations into equational specifications using graph products we illustrate the remaining translation (2) \rightarrow (3) from finite-tree automatic to VRS-equational presentations on graphs as we did for PR and RGTRS. That is, we build the terms of the presentation from the bottom up while also simulating the automata constituting the tree-automatic presentation by VRS-operations.

Start with a graph (V, E) that is definable via finite-subset interpretation in the full binary tree. By the fundamental correspondence that wMSO-definable relations in a regular tree are exactly those that are recognised by tree automata operating on finite trees, we see that V may be taken to be a regular set of finite Σ -labelled binary trees, and E is recognised by an automaton \mathcal{A} accepting pairs of such trees.

The tree automaton \mathcal{A} has transition rules (here we read them from left-to-right, i.e in top-down fashion, but that is a matter of choice and the simulation will actually proceed from bottom up) of the form

$$r : (q, \langle a, b \rangle, q_0, q_1) \quad \text{with } a, b \in \{0, 1, \square\}$$

where the symbol \square is necessary for padding either components of a pair of trees so that they have the same shape. It indicates the fact that no node is defined in the current position, i.e. that the automaton finds itself below a leaf of the respective tree (while still reading the other).

We may assume that the transition rules enforce a proper usage of the padding symbols.

We introduce edge relations E_q and E_r for each state q and each rule r of the automaton. The simulation of transitions of the synchronous automaton on *pairs of labelled trees* necessitates a more sophisticated recursion scheme associated to the following type definition of $\{0, 1\}$ -labelled binary branching trees.

$$\mathcal{T} = \perp \oplus (\{0, 1\} \otimes \mathcal{T} \otimes \mathcal{T}) \quad (1.6)$$

There is a natural identification of terms of this type and of those of the more natural type definition (1.4). As far as unary predicates are concerned the current type definition does not provide any advantage. However, compared with (1.4) the current type definition has a more powerful associated recursion scheme allowing for defining non-trivial *binary* relations between terms with different root labels. This will allow us to specify tree-automatic graphs via VRS-equations of the following form analogous to (1.6)

$$X = \vartheta (\bullet^\perp \oplus (\vartheta_0 \otimes_S \vartheta_1(X) \otimes_S \vartheta_2(X))) \quad (1.7)$$

Here too, as in (1.3) and in (1.5) the ϑ 's are VR-expressions facilitating the simulation of the automaton. The expression ϑ_0 specifies the graph with vertex set $\{0, 1\}$ and having an r -labelled edge from a to b for each rule r such that $r = (\cdot, \langle a, b \rangle, \cdot, \cdot)$ and with VR operations (here equivalently expressed as positive quantifier-free definable operations) responsible for updating the edge relations to simulate the transitions of \mathcal{A} . This is done in two phases.

- First, in preparation, state-labelled edges are used to ‘enable’ compatible rule-labelled edges in either copy of the graph: for each rule $r = (\cdot, \langle \cdot, \cdot \rangle, q_1, q_2)$ and $i \in \{1, 2\}$ the expression ϑ_i adds an E_r -edge from x to y for every E_{q_i} -edge from x to y in the graph.
- Then, after the synchronous product of rule-labelled edges has been taken, edges labelled by rules are renamed to their resulting states: ϑ adds for each state q an E_q -edge from x to y for every E_r -edge from x to y such that $r = (q, \langle \cdot, \cdot \rangle, \cdot, \cdot)$. In addition, ϑ deals with the case when either x or y is the singleton tree \perp . For this we may assume that all necessary information is coded in vertex labels implemented as reflexive edges and maintained along with the rest of the edge labels as explained here.

Finally, to obtain the graph G' as required in item (3) of Theorem 1.2.26

we also use vertex colours to keep track of the states of the tree automaton recognising V . The generalisation of this construction to arbitrary relational structures is straightforward.

1.3 Automatic Structures

1.3.1 Fundamentals

This section concerns structures with internal presentations consisting of automata operating synchronously on their inputs. The starting point of this investigation is the robust nature of finite automata. In particular, synchronous automata are effectively closed under certain operations that can be viewed in logical terms, i.e. Boolean operations, projection, cylindrification and permutation of arguments. Thus a structure whose domain and atomic operations are computable by such automata has decidable first-order theory (Definition 1.3.2 and Theorem 1.3.4).

Example 1.3.1 (i) The domain and relations of the following structure are regular.

$$\mathcal{S}_\Sigma = (\Sigma^*, \{\mathbf{suc}_a\}_{a \in \Sigma}, \prec_{\text{prefix}}, \mathbf{e1})$$

where Σ^* is the set of finite words over alphabet Σ , the binary relation \mathbf{suc}_a is the successor relation (x, xa) for $x \in \Sigma^*$, the binary relation \prec_{prefix} is the prefix relation and the binary relation $\mathbf{e1}$ is the equal-length relation.

(ii) The following structure can be coded (eg. in base k least significant digit first) so that the domain and atomic operations are regular.

$$\mathcal{N}_k = (\mathbb{N}, +, |_k)$$

where $+$ is the usual addition on natural numbers and $x |_k y$ holds precisely when x is a power of k and x divides y .

Actually the link between synchronous automata and logic goes both ways. It was first expressed in terms of weak monadic second-order logic: a set of tuples (A_1, \dots, A_n) of finite sets of natural numbers is weak monadic second-order definable in (\mathbb{N}, S) if and only if the corresponding n -ary relation of characteristic strings (a subset of $(\{0, 1\}^*)^n$) is synchronous rational. This was proved by [27] and [68], and is implicit in [135].

A first-order characterisation was provided by [65]: a relation $R \subset (\Sigma^*)^n$ is synchronous rational if and only if R is first-order definable

in \mathcal{S}_Σ for $|\Sigma| \geq 2$. Similarly, the Büchi-Bruyère Theorem states that a relation $R \subset \mathbb{N}^n$ (coded in base $k \geq 2$ least significant digit first) is synchronous rational if and only if it is first-order definable in \mathcal{N}_k (proofs of which can be found in [104] and [137]).

These results were generalised to full MSO on the line (\mathbb{N}, S) and weak MSO and full MSO on the tree $(\{0, 1\}^*, \text{suc}_0, \text{suc}_1)$ and form the basis of the logical characterisation of automatic structures (Section 1.3.4). However, we start with the more common internal definition.

Recall that the four basic types of automata operate on finite or infinite words or trees. So, let \square be one of `word`, `ω -word`, `tree`, `ω -tree`.

We consider a structure $\mathfrak{B} = (B, \{R_i\})$ comprising relations R_i over the domain $\text{dom}(\mathfrak{B}) = B$. Thus constants and operations are implicitly replaced by their graphs.

Definition 1.3.2 (Automatic presentation) A \square -automatic presentation of \mathfrak{B} consists of a tuple $\mathfrak{d} = (\mathcal{A}, \mathcal{A}_\approx, \{\mathcal{A}_i\})$ of finite synchronous \square -automata and a *naming function* $f : \mathcal{L}(\mathcal{A}) \rightarrow B$ such that

- Each $\mathcal{L}(\mathcal{A}_i)$ is a relation on the set $\mathcal{L}(\mathcal{A})$.
- $\mathcal{L}(\mathcal{A}_\approx)$ is a congruence relation on the structure $(\mathcal{L}(\mathcal{A}), \{\mathcal{L}(\mathcal{A}_i)\}_i)$.
- The quotient structure is isomorphic to \mathfrak{B} via f .

Moreover, the quotient structure is called an *automatic copy* of \mathfrak{B} . We say that the presentation is *injective* whenever f is, in which case \mathcal{A}_\approx can be omitted.

Definition 1.3.3 (Automatic structure¹⁰) A structure \mathfrak{B} is \square -*automatic* if it has an \square -automatic presentation. If \mathfrak{B} is \square -automatic for some \square then \mathfrak{B} is simply called *automatic*. The classes of automatic structures are respectively denoted by `S-AutStr`, `ω S-AutStr`, `T-AutStr` and `ω T-AutStr`.

The following theorem motivates the study of automatic structures and so may be called the *Fundamental Theorem* of automatic structures/presentations.

Theorem 1.3.4 (Definability) *There is an algorithm that given a \square -automatic presentation (\mathfrak{d}, f) of a structure \mathfrak{A} and a FO-formula $\varphi(\bar{x})$ in the signature of \mathfrak{A} defining a k -ary relation R over \mathfrak{A} , effectively constructs a synchronous \square -automaton recognising $f^{-1}(R)$.*

Immediate corollaries are

¹⁰ Some authors write *automatically presentable*.

- (i) *Decidability:* The FO-theory of every automatic structure is decidable.
- (ii) *Interpretations:* The class of \square -automatic structures is closed under FO-interpretations.

We point out that the Fundamental Theorem implies that every relation first-order definable from \square -regular relations is itself \square -regular.

Remark 1.3.5 One may allow finitely many parameters $\varphi(\bar{a}, \bar{x})$ under the following conditions. For finite-word and finite-tree presentations any parameters can be used. However, for ω -tree (and ω -word) presentations a parameter a can be used if $f^{-1}(a)$ contains a regular ω -tree (ultimately periodic ω -word).

Consequently \square -automatic structures (on a given signature) are closed with respect to operations such as disjoint union, ordered sum and direct product – each a special case of generalised products treated in [20, 23]. However **AutStr** and ω **S-AutStr** are not closed under weak direct-power. For instance, $(\mathbb{N}, +)$ is in **S-AutStr** but its weak direct-power is isomorphic to (\mathbb{N}, \times) , which is not in **S-AutStr** (see [20]). On the other hand, it is straightforward to see that **T-AutStr** and ω **T-AutStr** are closed under weak direct-power.

1.3.2 Examples

Obviously every finite structure is automatic. Here are some examples of structures with automatic presentations.

- Example 1.3.6** (Ordinals)
- (i) $(\omega, <) \in \mathbf{S-AutStr}$: The simplest automatic copy is the unary one: $(0^*, \{(0^k, 0^l) \mid k < l\})$.
 - (ii) Every ordinal below ω^ω is in **S-AutStr**: An automatic copy of ω^k is $((0^*1)^k, <_{\text{lex}})$ where $<_{\text{lex}}$ denotes the lexicographic order¹¹ which is clearly regular. In this presentation the naming function is

$$0^{n_{k-1}}1\dots0^{n_0}1 \mapsto n_{k-1}\omega^{k-1} + \dots + n_1\omega^1 + n_0 .$$

- (iii) Every ordinal below ω^{ω^ω} is in **T-AutStr**: recall that the ordinal ω^α has a representation as the set of functions $f : \alpha \rightarrow \omega$ with f equal to 0 in all but finitely many places. These functions are ordered as follows: $f < g$ if the largest β with $f(\beta) \neq g(\beta)$ has that $f(\beta) < g(\beta)$. Then for fixed k , a function $f : \omega^k \rightarrow \omega$ is coded by the tree T_f with

¹¹ Given an ordering on the symbols of the alphabet a word u is lexicographically smaller than w if either u is a proper prefix of w or if in the first position where u and w differ there is a smaller symbol in u than in w .

domain a finite subset of $0^*1^*2^*\cdots k^*$ so that for every β , expressed in Cantor-normal-form as $\omega^{k-1}c_0 + \omega^{k-2}c_1 + \cdots + \omega^0c_{k-1}$, $0 \leq c_i < \omega$, we have $T_f(0^{c_0}1^{c_1}\cdots(k-1)^{c_{k-1}}k^{f(\beta)}) = 1$.

Example 1.3.7 (Orderings) (i) $(\mathbb{Q}, <) \in \mathbf{S}\text{-AutStr}$: The countable linear order $(\{0, 1\}^*1, <_{\text{lex}})$ is dense without endpoints.

(ii) $(\mathbb{R}, <) \in \omega\mathbf{S}\text{-AutStr}$.

Example 1.3.8 (Groups) (i) Every finitely-generated group with an Abelian group of finite index is in $\mathbf{S}\text{-AutStr}$. And these are the only finitely generated word-automatic groups [116].

- (ii) The direct sum of countably many copies of $\mathbb{Z}/m\mathbb{Z}$ is in $\mathbf{S}\text{-AutStr}$.
- (iii) The subgroup $\mathbb{Z}[1/k]$ of rationals of the form $\{zk^{-i} \mid z \in \mathbb{Z}, i \in \mathbb{N}\}$ for fixed $k \in \mathbb{N}$ is in $\mathbf{S}\text{-AutStr}$.
- (iv) The Prüfer p -group $\mathbb{Z}(p^\infty) = \mathbb{Z}[1/p]/\mathbb{Z}$ (prime p) is in $\mathbf{S}\text{-AutStr}$ [114].
- (v) Real addition $(\mathbb{R}, +)$ is in $\omega\mathbf{S}\text{-AutStr}$.

However, the additive group of the rationals $(\mathbb{Q}, +)$ is not automatic [136]. In fact, Tsankov shows that no torsion free Abelian group that is p -divisible for infinitely many primes p is automatic.

Example 1.3.9 (Arithmetics) (i) $(\mathbb{N}, +)$ is in $\mathbf{S}\text{-AutStr}$: For every natural $k > 1$, the base k least-significant-digit-first presentation of naturals (with or without leading zeros) constitutes a naming function of an automatic presentation. A finite automaton can perform the school-book addition method while keeping track of the carry in its state. Such a presentation is injective when leading zeros are suppressed.

(ii) (\mathbb{N}, \cdot) is in $\mathbf{T}\text{-AutStr}$: The presentation is based on the unique factorisation of every natural number n into prime powers $2^{n_2}3^{n_3}\cdots p^{n_p}$. Each n_k is written, say in binary notation, on a single branch of a tree with domain 0^*1^* . Multiplication is reduced to the addition of corresponding exponents. This construction can naturally be generalised to give tree-automatic presentations of weak direct powers of word-automatic structures [20, 25].

Example 1.3.10 (Equivalence relations) The following have finite-word automatic presentations.

- (i) There is one class of size n for every $n \in \mathbb{N}$.
- (ii) There are $d(n)$ classes of size $n \in \mathbb{N}$ where $d(n)$ is the number of divisors of n . (This is the direct product of the previous equivalence relation with itself).

Example 1.3.11 (Free algebras) (i) The free algebra with n unary operations and at most ω many constants is in $\mathbf{S}\text{-AutStr}$.

- (ii) The free monoid generated by a single constant is in $\mathbf{S}\text{-AutStr}$. However, no non-unary free or even free-associative algebra on two or more constants is in $\mathbf{S}\text{-AutStr}$.
- (iii) The free algebra generated by countably many constants and any finite number of operations is in $\mathbf{T}\text{-AutStr}$.¹² For instance suppose there is one binary operation F . The domain of the presentation consists of all $\{F, c, \perp\}$ -labelled binary trees. The operation (representing F) takes trees S and T as input and returns the tree with domain the prefix-closure of $(\text{dom}(S) \cup \text{dom}(T))\{0, 1\}$ and taking the following values: the root position is labelled F ; position $\alpha 0$ is labelled by the label of S at position α ; position $\alpha 1$ by the label of T at position α (if either of these latter positions does not exist, the label is \perp). It is not known whether finitely generated (non-unary) term algebras are in $\mathbf{T}\text{-AutStr}$.

Example 1.3.12 (Boolean Algebras) The signature we work in consists of the symbols for boolean operations \cap, \cup, \cdot^c and constants \perp, \top .

- (i) Every finite power of the algebra of finite and co-finite subsets of \mathbb{N} is in $\mathbf{S}\text{-AutStr}$.
- (ii) The countable atomless Boolean algebra is in $\mathbf{T}\text{-AutStr}$: It is isomorphic to the algebra of sets consisting of the clopen sets in Cantor space. Each clopen set has a natural representation as a finite tree.
- (iii) The algebra of all subsets of \mathbb{N} is in $\omega\mathbf{S}\text{-AutStr}$.
- (iv) The algebra of all subsets of \mathbb{N} factored by the congruence of having finite symmetric difference is in $\omega\mathbf{S}\text{-AutStr}$. It is unknown whether this structure can be injectively presented in $\omega\mathbf{S}\text{-AutStr}$.
- (v) The interval algebra of the real interval $[0, 1]$ is in $\omega\mathbf{T}\text{-AutStr}$.
- (vi) The algebra of all subsets of $\{0, 1\}^*$ with a distinguished set \mathcal{F} consisting of those $X \subset \{0, 1\}^*$ such that for every path $\pi \in \{0, 1\}^\omega$ only finitely many prefixes of π are in X .

Example 1.3.13 (Graphs) (i) The infinite upright grid is in $\mathbf{S}\text{-AutStr}$: Here the structure is $(\mathbb{N} \times \mathbb{N}, \text{Up}, \text{Right})$ with the functions $\text{Right} : (n, m) \mapsto (n + 1, m)$ and $\text{Up} : (n, m) \mapsto (n, m + 1)$. It can be automatically presented on the domain a^*b^* with relations

$$R = \begin{pmatrix} a \\ a \end{pmatrix}^* \begin{pmatrix} b \\ a \end{pmatrix} \begin{pmatrix} b \\ b \end{pmatrix}^* \begin{pmatrix} \square \\ b \end{pmatrix}$$

¹² Communicated by Damian Niwinski.

and U defined by a similar regular expression.

- (ii) The transition graphs of pushdown automata are in $\mathbf{S}\text{-AutStr}$.¹³ Given a pushdown automaton \mathcal{A} with states Q , stack alphabet Γ , input alphabet Σ and transition relation Δ we can construct an automatic presentation of the transition graph of its configurations as follows. We take $Q\Gamma^*$ to be the domain of the presentation in which $q\gamma$ represents the configuration of state q and stack $\gamma \in \Gamma^*$. For each $a \in \Sigma$ there is an a -transition from $q\gamma$ to $q'\gamma'$ if, and only if, $\gamma = z\alpha$, $\gamma' = w\alpha$ and $(q, z, q', w) \in \Delta$ for some $z \in \Gamma$ and $w \in \Gamma^*$. Since Δ is finite, this relation is obviously regular for each a . Notice that in these presentations the transition relations are not only regular but in fact defined by prefix-rewriting rules (cf. Section 1.2.1 on context-free graphs).
- (iii) The transition graphs of Turing machines are in $\mathbf{S}\text{-AutStr}$ [87]. We can give an automatic presentation of each TM \mathcal{M} similar to those of pushdown automata. Configurations are encoded as strings $\alpha q \beta \in \Gamma^* Q \Gamma^*$ where α and β are the tape contents to the left, respectively, to the right of the head of \mathcal{M} , and q is the current state. Observe that, as opposed to presentations of pushdown graphs, the state is now positioned not at the left of the string but at the location of the head. Consequently, rewriting is not confined to prefixes, but rather occurs around the state symbol: transitions are of the form $\alpha u q w \beta \mapsto \alpha u' q' w' \beta$ for adequate u, w, u', w' and q, q' as determined by the transition function of \mathcal{M} . The fact that TM graphs are presentable using *infix rewriting* has the profound consequence that reachability questions in infix-rewriting systems are generally undecidable, as opposed to graphs of *prefix-rewriting* systems, whose monadic second-order theory is decidable (cf. Theorem 1.2.4).

Example 1.3.14 (Automata-theoretic structures) The following structures turn out to be universal for their respective classes (see Theorem 1.3.17).

- (i) Let

$$\mathcal{S}_\Sigma = (\Sigma^*, \{\mathsf{suc}_a\}_{a \in \Sigma}, \prec_{\mathsf{prefix}}, \mathsf{el})$$

and

$$\mathcal{S}_\Sigma^\omega = (\Sigma^{\leq\omega}, \{\mathsf{suc}_a\}_{a \in \Sigma}, \prec_{\mathsf{prefix}}, \mathsf{el})$$

¹³ For *visibly pushdown automata* the same representation of configurations also allows for the trace equivalence relation to be recognised by a finite automaton. In [10] this presentation was utilised to obtain a decidability result.

be the structures defined on finite, respectively on finite and ω -words, comprising the successor relations $\mathbf{suc}_a = \{(w, wa) \mid w \in \Sigma^*\}$; the prefix relation $u \prec_{\text{prefix}} w$ (where u is finite and w is finite or infinite); and the equal-length relation: $u \in w$ if, and only if, $|u| = |w|$. Clearly $\mathcal{S}_\Sigma \in \mathbf{S}\text{-AutStr}$ and $\mathcal{S}_\Sigma^\omega \in \omega\mathbf{S}\text{-AutStr}$. Note that if Σ is unary, then \mathcal{S}_Σ reduces to $(\mathbb{N}, +1, <, =)$.

- (ii) The structure $\mathcal{T}_\Sigma \in \mathbf{T}\text{-AutStr}$ has domain consisting of all finite binary Σ -labelled trees and has operations

$$(\preceq_{\text{ext}}, \equiv_{\text{dom}}, (\mathbf{suc}_a^d)_{d \in \{l, r\}, a \in \Sigma}, (\epsilon_a)_{a \in \Sigma})$$

where $T \preceq_{\text{ext}} S$ if $\text{dom}(T) \subset \text{dom}(S)$ and $S(\alpha) = T(\alpha)$ for $\alpha \in \text{dom}(T)$; $T \equiv_{\text{dom}} S$ if $\text{dom}(T) = \text{dom}(S)$; $\mathbf{suc}_a^d(T) = S$ if S is formed from T by extending its leaves in direction d and labeling each new such node by a ; and ϵ_a is the tree with a single node labelled a .

Similarly the structure $\mathcal{T}_\Sigma^\omega \in \omega\mathbf{T}\text{-AutStr}$ has domain consisting of all finite and infinite trees and operations

$$(\preceq_{\text{ext}}, \equiv_{\text{dom}}, (\mathbf{suc}_a^d)_{d \in \{l, r\}, a \in \Sigma}, (\epsilon_a)_{a \in \Sigma}).$$

that are restricted to finite trees, except that $T \preceq_{\text{ext}} S$ is defined as above but allows S to be an infinite tree.

1.3.3 Injectivity

Recall that an automatic presentation is injective if the naming function is injective. The problem of injectivity is this:

Does every \square -automatic structure have an injective \square -automatic presentation?

An injective presentation has the advantage that it is easier to express certain cardinality-properties of sets of elements (Theorem 1.4.6). We consider the four cases.

Finite words

From a finite-word automatic presentation of \mathfrak{A} one defines an injective presentation of \mathfrak{A} by restricting to a regular set D of unique representatives. These can be chosen using a regular well-ordering of the set of all finite words. For instance, define $D \subset \mathcal{L}(\mathcal{A})$ to be the length-lexicographically least words from each $\mathcal{L}(\mathcal{A}_{\approx})$ equivalence class.

Finite trees

Except in the finite word case, there is no regular well ordering of the set of all finite trees [39]. However one can still convert a finite-tree automatic presentation into an injective one [47]. The idea is to associate with each tree t a new tree \hat{t} of the following form: the domain is the intersection of the prefix-closures of the domains of all trees that are $\mathcal{L}(\mathcal{A}_\approx)$ -equivalent to t ; a node is labelled σ if t had label σ in that position; a leaf x is additionally labelled by those states q from which the automaton \mathcal{A}_\approx accepts the pair consisting of the subtree of t rooted at x and the tree with empty domain.¹⁴ Using transitivity and symmetry of $\mathcal{L}(\mathcal{A}_\approx)$, if $\hat{t} = \hat{s}$ then t is $\mathcal{L}(\mathcal{A}_\approx)$ -equivalent to s . Moreover each equivalence class is associated with finitely many new trees, and so a representative may be chosen using any fixed regular linear ordering of the set of all finite trees.

ω -words

There is a structure in $\omega\text{-AutStr}$ that does not have an injective ω -word automatic presentation [82]. The proof actually shows that the structure has no injective presentation in which the domain and atomic relations are Borel.

However, every *countable* structure in $\omega\text{-AutStr}$ does have an injective ω -word automatic presentation [85] (and consequently is also in S-AutStr). This follows from the more general result that every ω -word regular equivalence relation with countable index has a regular set of representatives [85].

ω -trees

It has not yet been settled whether injective presentations suffice, even for the countable structures.

1.3.4 Alternative characterisations

Automatic structures were defined internally. We now present equivalent characterisations: logical (FO and MSO) and equational.

First-order characterisations

In order to capture regularity in the binary representation of \mathbb{N} using first-order logic Büchi suggested the expansion $(\mathbb{N}, +, \{2^n \mid n \in \mathbb{N}\})$ of

¹⁴ The construction given in [47] is slightly more general and allows one to effectively factor finite-subset interpretations in any tree.

Presburger arithmetic, which is, however, insufficient (see [26]). Boffa and Bruy  re considered expressively complete expansions of $(\mathbb{N}, +)$ by relations of the form $x \mid_k y$ (defined to hold precisely when x is a power of k and x divides y).

Theorem 1.3.15 (Büchi-Bruyére, cf. [26]) A relation $R \subseteq \mathbb{N}^r$ is regular in the least-significant-digit-first base k presentation of \mathbb{N} if, and only if, R is first-order definable in the structure $\mathcal{N}_k = (\mathbb{N}, +, |_k)$.

Closer to automata, the structures \mathcal{S}_Σ on words (see example 1.3.14) allow one to define every regular relation on alphabet Σ .

Theorem 1.3.16 ([65]) Let Σ be a finite, non-unary alphabet. A relation over Σ^* is regular if, and only if, it is first-order definable in \mathcal{S}_Σ .

The proofs of these theorems are by now standard. From left to write one writes a formula $\phi_{\mathcal{A}}(x)$ that expresses the existence of a successful run in automaton \mathcal{A} on input x . For the other direction the atomic operations of the structures are regular forms the base case for structural induction on the formula. Both theorems transfer to automatic structures by replacing definability with interpretability [24, 25].

Theorem 1.3.17 (First-order characterisation of $\mathbf{S}\text{-AutStr}$) *The following conditions are equivalent.*

- $\mathfrak{A} \in S\text{-AutStr}$.
 - \mathfrak{A} is first-order interpretable in S_Σ (for some/all Σ with $|\Sigma| \geq 2$).
 - \mathfrak{A} is first-order interpretable in N_k (for some/all $k \geq 2$).

These structures have been called *universal* or *complete* (with respect to FO-interpretations) for the class of finite-word automatic structures. There are similar universal structures for the other classes of automatic structures. These are the structures S_Σ^ω , \mathcal{T}_Σ and $\mathcal{T}_\Sigma^\omega$ from Example 1.3.14 [20, 14].

Finite set interpretations

The four notions of automatic presentation have straightforward reformulations in terms of subset interpretations either in the line $\Delta_1 = (\mathbb{N}, \text{suc})$ or in the tree $\Delta_2 = (\{0, 1\}^*, \text{suc}_0, \text{suc}_1)$.

Theorem 1.3.18 (Automatic presentations as subset interpretations)
There are effective transformations establishing the following equivalences.

- (i) $\mathfrak{A} \in \mathbf{S}\text{-AutStr}$ if, and only if, $\mathfrak{A} \leq_{\text{fset}} \Delta_1$
- (ii) $\mathfrak{A} \in \omega\mathbf{S}\text{-AutStr}$ if, and only if, $\mathfrak{A} \leq_{\text{set}} \Delta_1$
- (iii) $\mathfrak{A} \in \mathbf{T}\text{-AutStr}$ if, and only if, $\mathfrak{A} \leq_{\text{fset}} \Delta_2$
- (iv) $\mathfrak{A} \in \omega\mathbf{T}\text{-AutStr}$ if, and only if, $\mathfrak{A} \leq_{\text{set}} \Delta_2$

Equivalently, one may formulate universality with respect to FO interpretations. Following [47] we define the (*finite*) *subset envelope* $\mathcal{P}_{(f)}(\mathfrak{A})$ of a structure \mathfrak{A} by adjoining to \mathfrak{A} its (finite) subsets as new elements ordered by set inclusion.

Definition 1.3.19 Given $\mathfrak{A} = (A, \{R_i\})$ write $P(A)$ for the set of all subsets of A . The *subset envelope* $\mathcal{P}(\mathfrak{A})$ is the structure with domain $P(A)$ and relations $R'_i := \{\{\{a_1\}, \dots, \{a_n\}\} \mid (a_1, \dots, a_n) \in R_i\}$ and the subset relation \subseteq defined on $P(A)$. The *finite-subset envelope* $\mathcal{P}_f(\mathfrak{A})$ is the substructure of $\mathcal{P}(\mathfrak{A})$ whose domain is the set of finite subsets of A .

It is immediate that

$$\mathfrak{B} \leq_{(f)\text{set}} \mathfrak{A} \iff \mathfrak{B} \leq_{\text{FO}} \mathcal{P}_{(f)}(\mathfrak{A})$$

In particular, this yields natural universal structures, with respect to FO-interpretations, for each of the four classes of automatic structures.

- Corollary 1.3.20**
- (i) $\mathcal{P}_f(\Delta_1)$ is universal for $\mathbf{S}\text{-AutStr}$.
 - (ii) $\mathcal{P}(\Delta_1)$ is universal for $\omega\mathbf{S}\text{-AutStr}$.
 - (iii) $\mathcal{P}_f(\Delta_2)$ is universal for $\mathbf{T}\text{-AutStr}$.
 - (iv) $\mathcal{P}(\Delta_2)$ is universal for $\omega\mathbf{T}\text{-AutStr}$.

VRS-Equational structures

Recall that the VRS-algebra of graphs extends the VR-algebra with the synchronous product operation and that VRS-equational systems define exactly the finite-tree automatic graphs (see Section 1.2.4 and Theorem 1.2.26).

A finite VRS-equational system whose unfolding is a linear VRS-term specifies a structure in $\mathbf{S}\text{-AutStr}$. This happens if in the defining equations one of the arguments of each occurrence of \oplus and of \otimes_S is a finite graph (and so these act like unary operations). Conversely, for word-automatic presentations Equation (1.7) reduces to the following form:

$$X = \vartheta(\bullet^\perp \oplus (\vartheta_0 \otimes \vartheta_1(X))) \quad (1.8)$$

This scheme matches the following type definition obtained by restricting (1.6) to words:

$$\mathcal{T} = \perp \oplus (\{0, 1\} \otimes \mathcal{T}) \quad (1.9)$$

This recursive definition of the set of words has the same advantage over (1.2) as (1.6) has over (1.4) when it comes to defining binary relations over words via structural induction, e.g. via finite automata. Over words we have the following special case of Theorem 1.2.26.

Theorem 1.3.21 (Colcombet [50])

For every countable structure \mathfrak{A} the following are equivalent

- (1) \mathfrak{A} is isomorphic to a word-automatic graph.
- (2) \mathfrak{A} is the restriction of some \mathfrak{B} to its elements of a certain colour, where \mathfrak{B} can be specified by a VR-equation $Z = \pi(X)$, where π simply forgets some of the structure of X , together with a VRS-equation for X of the form (1.8);
- (3) \mathfrak{A} is finite-subset interpretable in (\mathbb{N}, suc) .

The equivalence of the first and the third item is a direct consequence of the classical correspondence of automata on words and monadic second-order logic of one successor and was already stated in Theorem 1.3.18. Nonetheless, this can also be inferred from the fact that the solution term obtained by unfolding (1.8) is (essentially) a periodic linear VRS-term that evaluates, via a finite-subset interpretation, to the word-automatic structure specified by equation (1.8).

More generally, let VRS^- denote the extension of VR with unary operations $X \mapsto G_0 \otimes_S X$ where G_0 is any finite graph. Moreover let us call a *chain interpretation* a subset interpretation in a tree where each of the subsets representing an element is linearly ordered by the ancestor relation of the tree. It is not hard to see that solutions of finite systems of VRS^- -equations are finite-chain interpretable in a regular tree and that these in turn are word automatic [50].

1.3.5 Rational graphs

If we allow the more general *asynchronous automata* in the definition of an automatic presentation of a graph we get the notion of a rational graph. Thus vertices are labelled with finite words of a rational language over some finite alphabet Σ , and the edge relations are required to be rational subsets of $\Sigma^* \times \Sigma^*$.

With no aim for completeness we list below some results on rational graphs (asynchronous) in comparison with automatic graphs (synchronous). For a comprehensive treatment the reader is referred to [105].

The class of rational graphs strictly includes that of finite-word automatic graphs. In their seminal paper [87] Khoussainov and Nerode also introduced asynchronous automatic structures. As an example they gave an asynchronous automatic presentation of ω^ω , which is not in **S-AutStr** (see Theorem 1.4.12). Asynchronous automatic presentations of Cayley-graphs of finitely generated groups have also been considered as generalisations of ‘automatic groups’ [31].

The price of increasing expressiveness is a loss of tractability: in general, rational graphs do not have a decidable first-order theory. This renders rational graphs useless for representing data, let alone programs. However, in the context of formal language theory rational graphs seem to fill a gap. Considering rational graphs as infinite automata, i.e. as acceptors of languages, Morvan and Stirling have shown that they trace exactly the context-sensitive languages [108, 107] (see also [34] for a simplified approach). Rispal and others [123, 107, 34] have subsequently observed that this holds true for automatic graphs as well.

Although first-order queries on rational graphs are in general intractable there are some interesting decidable subclasses.

Morvan observed that by a result of Eilenberg and Schützenberger, graphs defined by rational relations over a *commutative* monoid have a decidable first-order theory. In particular, over the unary alphabet the monoid structure is isomorphic to $(\mathbb{N}, +)$ whence the unary rational graphs are those first-order definable in $(\mathbb{N}, +)$ [105]. Similarly, rational graphs over $(\mathbb{N}, +)^d$ are those having a d -dimensional first-order interpretation in $(\mathbb{N}, +)$.

Carayol and Morvan showed that on rational graphs that also happen to be trees (this is an undecidable property) first-order logic is decidable [36, 106]. The decision method is based on locality of FO as formulated by Gaifman and uses a compositional technique. The authors also exhibit a rational graph that is a finitely branching tree but is not finite-word automatic.

1.3.6 Generalisations

Automata with oracles

Consider an expansion Δ_i^O of $\Delta_i := ([i]^*, \text{suc}_0, \dots, \text{suc}_{i-1})$ by a unary predicate $O \subset [i]^*$. Every MSO formula (with free MSO variables) of the expanded structure corresponds to a tree automaton with *oracle* O . An automaton with oracle is one that, while in position $u \in [i]^*$, can decide on its next state using the additional information of whether or

not $u \in O$. Thus for automata working on infinite words/trees the oracle O is simply read as part of the input. In the case of automata working on finite words/trees, the entire oracle is scanned, and so the acceptance condition should be taken appropriately (eg. Muller/Rabin).

Call a set O *decidable* if $\text{MSO}(\Delta_i^O)$ is decidable, and *weakly decidable* if $\text{wMSO}(\Delta_i^O)$ is decidable. Early work on decidable oracles used the contraction method to show that certain oracles on the line, such as $\{n! \mid n \in \mathbb{N}\}$, are decidable [67]. This was extended to the profinitely ultimately periodic words [38], which it turns out capture all the decidable unary predicates on the line [119, 120]. Nonetheless, it is still of interest to produce explicit examples of decidable oracles, see for instance [38, 74, 75, 7].

Definition 1.3.22 If in the definition of automatic presentation (1.3.2) we replace \square -automata with \square -automata with oracle O , we get a notion of \square -automatic presentation with oracle O . A structure is called *automatic with oracle* if it has a \square -automatic presentation with some oracle.

Example 1.3.23 The group of rationals $(\mathbb{Q}, +)$ has recently been shown to have no word-automatic presentation [136]. However it is finite-word automatic with oracle $\#2\#3\#4\dots$. This is based on the idea, independently found by Frank Stephan and Joe Miller and reported in [114], that there is a presentation of $([0, 1] \cap \mathbb{Q}, +)$ by finite words in which $+$ is regular, but the domain is not: every rational in $[0, 1]$ can be expressed as $\sum_{i=2}^n \frac{a_i}{i!}$ for a unique sequence of natural numbers a_i satisfying $0 \leq a_i < i$. The presentation codes this rational as $\#a_2\#a_3\#a_4\dots$ where a_i is written in decimal notation (and hence has length less than the length of i written in decimal notation). Addition is performed with the least significant digit first, based on the fact that

$$\frac{a_i + b_i + c}{i!} = \frac{1}{(i-1)!} + \frac{a_i + b_i + c - i}{i!}$$

where $c \in \{0, 1\}$ is the carry in.

We immediately have that a structure is (finite-)word/tree automatic with oracle O if and only if it is (finite) set interpretable in Δ_1^O/Δ_2^O . Hence we have the following generalisation of the Fundamental Theorem and its corollaries (1.3.4).

Theorem 1.3.24 (i) Definability: *Say (\mathfrak{d}, f) is a \square -automatic presentation with oracle O of a structure \mathfrak{A} and $\varphi(\bar{x})$ is a FO-formula in the*

signature of \mathfrak{A} defining a k -ary relation R over \mathfrak{A} . Then the relation $f^{-1}(R)$ is recognised by an \square -automaton with oracle O .

- (ii) *Interpretations: The class of \square -automatic structures with oracle O is closed under FO-interpretations.*
- (iii) *Decidability: The previous statements can be made effective under the following conditions.*

- 1 *For $\square \in \{\text{word, tree}\}$ we require that $\text{wMSO}(\Delta_i^O)$ be decidable.*
- 2 *For $\square \in \{\omega\text{-word, } \omega\text{-tree}\}$ we require that $\text{MSO}(\Delta_i^O)$ be decidable.*

In particular, under these conditions, every \mathfrak{A} that is \square -automatic with oracle O has decidable FO-theory.

Of course Δ_i^O can be viewed as a coloured tree. As in Corollary 1.3.20 we have universal structures with respect to FO-definability. For instance $\mathcal{P}(\Delta_2^O)$ is universal for $\omega\text{T-AutStr}$ with oracle O . The following result concerns finite-set interpretations in arbitrary trees.

Theorem 1.3.25 ([47]) *To every finite set interpretation \mathcal{I} one can effectively associate a wMSO interpretation \mathcal{J} such that for every tree t and structure \mathfrak{A} if $\mathcal{P}_f(\mathfrak{A}) \cong \mathcal{I}(t)$ then $\mathfrak{A} \cong \mathcal{J}(t)$.*

This can be used to show that certain structures, such as the random graph, are not finite-tree automatic in the presence of any oracle [47].

1.3.7 Subclasses

In this section we restrict the complexity of the regular domains in automatic presentations to yield some of the more robust subclasses of S-AutStr and T-AutStr .

Polynomial domain

The most natural restriction is to consider presentations where the words and trees take labels from a unary alphabet $|\Sigma| = 1$. Word-automatic presentations over a unary alphabet were introduced and studied by Blumensath [20] and Rubin [89, 124].

The *density* of a language $L \subset \Sigma^*$ is the function $n \mapsto |L \cap \Sigma^n|$.

Definition 1.3.26 A structure is *unary automatic* if it has an injective word-automatic presentation in which the domain consists of words from a unary alphabet. A structure is *p-automatic* if it has an injective word-automatic presentation in which the domain has polynomial density. Let 1-AutStr and P-AutStr denote these respective classes of structures.

Regular sets of polynomial density were characterised by Szilard et al. [131] as being a finite union of the form

$$D = \bigcup_{i < N} u_{i,1} v_{i,1}^* u_{i,2} \dots u_{i,n_i} v_{i,n_i}^* u_{i,n_i+1} \quad (1.10)$$

where the degree of the polynomial of the density function is equal to the maximum of the n_i 's. In [6] it was demonstrated that every finite-word-automatic presentation over a domain as in (1.10) can be transformed into an equivalent one (cf. Section 1.4.4) over a domain that is a regular subset of

$$a_1^* a_2^* \dots a_n^*$$

where n is equal to the maximum of the n_i 's. In particular, word-automatic presentations over a domain of linear density are unary automatic. This transformation yields a kind of normal-form of word-automatic presentations over a polynomially growing domain.

Theorem 1.3.27 ([6]) *A structure \mathfrak{A} has an automatic presentation over a domain of density $\mathcal{O}(n^d)$ if, and only if, it has a d -dimensional interpretation in $\mathfrak{M} := (\mathbb{N}, <, \{\equiv_{(mod m)}\}_{m > 1})$ if, and only if, it is finite-subset interpretable in $\Delta_1 := (\mathbb{N}, \text{suc})$ with subsets of size at most d .*

Corollary 1.3.28 ([113],[20]) *A structure \mathfrak{A} is unary automatic if, and only if, it is first-order definable in \mathfrak{M} if, and only if, it is MSO-interpretable in Δ_1 .*

Unary automatic structures form a very restricted subclass of VR-equational structures and have a decidable MSO-theory. Using pumping arguments one can show that Presburger arithmetic $(\mathbb{N}, +)$ has no p-automatic presentation [20, 121]. On the other hand, the infinite grid is p-automatic but not unary automatic. Thus we have

$$\text{1-AutStr} \subsetneq \text{P-AutStr} \subsetneq \text{S-AutStr}.$$

The expansion of \mathfrak{M} with the successor function **suc** and a constant for 0 admits quantifier elimination. Hence, every p-automatic structure can be interpreted in $(\mathbb{N}, 0, \text{suc}, <, \{\equiv_{(mod m)}\}_{m > 1})$ using quantifier-free formulas.

Every p-automatic structure inherits the PSPACE upper-bound on the complexity of its first-order theory from \mathfrak{M} . This is as low as possible since FO model-checking is PSPACE-hard for any structure with at least two elements. Adding even the simplest form of iteration to

FO leads to undecidability. For every k -counter machine it is straightforward to construct a p-automatic presentation of its configuration graph where each configuration (q, n_1, \dots, n_k) is represented by the word $qc_1^{n_1} \cdots c_k^{n_k}$. It follows that the first-order theory with reachability $\text{FO}[\mathbf{R}]$ of a p-automatic structure is undecidable in general. In comparison, while unary automatic structures have a decidable MSO-theory, the $\text{FO}(\text{DTC})$ theory of $(\mathbb{N}, \text{succ})$ interprets full first-order arithmetic and is therefore highly undecidable [20].

Observe, that graphs having rational presentation over a finitely generated commutative monoid (cf. Section 1.3.5) can be seen as analogues of p-automatic graphs. Indeed, every monoid element is represented by some word $g_1^{r_1} g_2^{r_2} \cdots g_n^{r_n}$ over the generators.

Finite-rank tree-automatic presentations

The analogue of p -automatic to tree-automatic structures is restricting to presentations involving trees of bounded rank. Intuitively the rank of a tree corresponds to its branching degree (which can be measured in terms of the Cantor-Bendixson rank).

Recall a Σ -labelled n -ary tree T is a function from a prefix-closed subset of $[n]^*$ to Σ . We say that T has *rank* k if its domain has polynomial density of degree at most k .

A finite-tree automatic presentation is called of *rank* k if for some regular language D of polynomial density of degree at most k the domain of every tree in the presentation is a subset of D . Collectively we speak of *bounded-rank tree-automatic presentations*. The class of structures with rank k presentations is denoted $\mathbf{k-T-AutStr}$.

Example 1.3.29 The ordinal ω^{ω^k} has a rank $k + 1$ tree-automatic presentation.

Let \mathcal{T}_k denote the structure corresponding to the unlabelled k -ary tree with domain $0^*1^*\cdots(k-1)^*$. Note that \mathcal{T}_k is wMSO-interpretable in the ordinal ω^k (in the signature of order), and vice-versa.

Proposition 1.3.30 *The following are equivalent.*

- \mathfrak{A} is in $\mathbf{k-T-AutStr}$,
- \mathfrak{A} is finite-set interpretable in \mathcal{T}_k (or equivalently in the ordinal ω^k),
- \mathfrak{A} is the solution of a finite system of VRS-equations whose unfolding is a term-tree of rank k .

The hierarchy is strict:

$$\mathbf{S}\text{-AutStr} = \mathbf{1}\text{-T-AutStr} \subsetneq \mathbf{2}\text{-T-AutStr} \subsetneq \dots \subsetneq \mathbf{T}\text{-AutStr}.$$

Indeed, if $\mathbf{k+1}\text{-T-AutStr} = \mathbf{k}\text{-T-AutStr}$ for some k then the finite-subset envelope $\mathcal{P}_f(\omega^{k+1})$ would be finite-set interpretable in ω^k . But by Theorem 1.3.25 then ω^{k+1} is wMSO interpretable in ω^k , which is known not to be possible [98, Lemma 4.5].¹⁵

1.3.8 Comparison of classes

Since words are special cases of trees, and finite ones special cases of infinite ones, one immediately sees the inclusions indicated by the arrows in the figure. All the arrows except for the dotted one are known to be strict inclusions. We now discuss the separating examples as well as the double lines indicating equality of the classes when restricted to countable structures. Since $\omega\mathbf{S}\text{-AutStr}$ and $\omega\mathbf{T}\text{-AutStr}$ contain uncountable structures while $\mathbf{S}\text{-AutStr}$ and $\mathbf{T}\text{-AutStr}$ do not, we split our discussion along these lines.

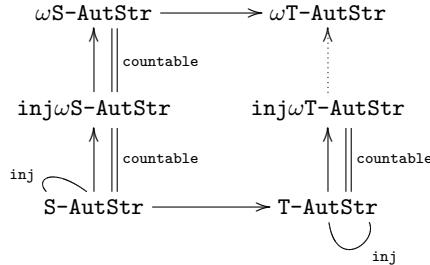


Figure 1.2 Relationship of classes of automatic structures

Countable structures

The structure (\mathbb{N}, \times) separates $\mathbf{T}\text{-AutStr}$ from $\mathbf{S}\text{-AutStr}$ (see [20], or [88] for an alternative proof).

Every injective $\omega\mathbf{S}\text{-AutStr}$ presentation of a countable structure can be effectively transformed into a $\mathbf{S}\text{-AutStr}$ presentation. This is because a countable ω -regular set $X \subseteq \{0, 1\}^\omega$ only contains ultimately periodic words, and moreover there is a bound on the size of the periods

¹⁵ We thank Łukasz Kaiser for discussions on the notions of this section and Alex Rabinovich for providing the latter reference.

(which can be computed from an automaton for X). Similar facts hold for countable regular sets of infinite trees [115].

The next theorem generalises this in the word case:

- Theorem 1.3.31** ([85]) (i) *The countable structures in $\omega\text{S-AutStr}$ are precisely those in S-AutStr .*
(ii) *Given a (not necessarily injective) automatic presentation of some $\mathfrak{A} \in \omega\text{S-AutStr}$ it is decidable whether \mathfrak{A} is countable or not, and if it is, an automatic presentation of \mathfrak{A} over finite words can be constructed.*

On the other hand, we do not know whether every countable structure in $\omega\text{T-AutStr}$ is in T-AutStr .

Uncountable structures

The only known non-trivial methods dealing with uncountable structures appear in [82]:

- (i) The algebra $(\mathcal{P}(\{0, 1\}^*), \cap, \cup, \cdot^c, \mathcal{F})$ from example 1.3.12(6) is an uncountable structure separating $\omega\text{T-AutStr}$ from $\omega\text{S-AutStr}$.
- (ii) Recall Example 1.3.12(4) consisting of the algebra of subsets of \mathbb{N} (call it \mathcal{A}) quotiented by having finite symmetric difference (call it \approx). Construct a variant structure as the disjoint union of \mathcal{A} and \mathcal{A}/\approx , with a unary predicate U identifying the elements of \mathcal{A} and a binary relation R relating $a \in A$ to its representative in \mathcal{A}/\approx . This uncountable structure separates $\omega\text{S-AutStr}$ from $\text{inj}\omega\text{S-AutStr}$.

1.4 More on word-automatic presentations

1.4.1 Beyond first-order logic

The Fundamental Theorem can be strengthened to include order-invariant definable formulas as well as certain additional quantifiers.

Generalised quantifiers

We briefly recall the definition of generalised quantifiers as introduced by Lindström.

Definition 1.4.1 Fix a finite signature $\tau = (R_i)_{i \leq k}$, where R_i has associated arity r_i . A *quantifier* Q is a class of τ -structures closed under isomorphism. Let σ be another signature. Given σ -formulas $\Psi_i(\bar{x}_i, \bar{z})$

with $|\bar{x}_i| = r_i$ ($i \leq k$), the syntax $Q\bar{x}_1, \dots, \bar{x}_n(\Psi_1, \dots, \Psi_k)$ has the following meaning on a σ -structure \mathcal{A} :

$$(\mathcal{A}, \bar{a}) \models Q\bar{x}_1, \dots, \bar{x}_k(\Psi_1, \dots, \Psi_k) \text{ iff } (A; \Psi_1^{\mathcal{A}}(\cdot, \bar{a}), \dots, \Psi_k^{\mathcal{A}}(\cdot, \bar{a})) \in Q,$$

where $\Psi^{\mathcal{A}}(\cdot, \bar{a})$ is the relation defined in \mathcal{A} by Ψ with parameters \bar{a} . The *arity* of a quantifier is the maximum of the r_i s. A quantifier is n -ary if its arity is at most n .

The extension of first-order logic by a collection \mathbf{Q} of generalised quantifiers will be denoted $\text{FO}[\mathbf{Q}]$.

Examples 1.4.2 (i) The unary quantifier $\{(A; X) \mid \emptyset \neq X \subset A\}$ is ‘there exists’.

- (ii) The unary quantifier ‘there exists infinitely many’, written \exists^∞ , is the class of structures $(A; X)$ where X is an infinite subset of A .
- (iii) The unary *modulo quantifier* ‘there are k modulo m many’ (here $0 \leq k < m$), written $\exists^{(k,m)}$, is the class of structures $(A; X)$ where X contains k modulo m many elements. Write \exists^{mod} for the collection of modulo quantifiers.
- (iv) The unary *Härtig quantifier* is the class of structures $(A; P, Q)$ where $P, Q \subset A$ and $|P| = |Q|$.
- (v) Every set $C \subset (\mathbb{N} \cup \{\infty\})^n$ induces the unary *cardinality quantifier* $Q_C = \{(A; P_1, \dots, P_n) \mid (|P_1|, \dots, |P_n|) \in C\}$. In fact, a given unary quantifier over signature $(R_i)_{i \leq k}$ is identical to some cardinality quantifier with $n = 2^k$.
- (vi) The binary *reachability quantifier* is the class of structures of the form $(A; E, \{c_s\}, \{c_f\})$ where $E \subset A^2$, $c_s, c_f \in A$, and there is a path in the directed graph $(A; E)$ from c_s to c_f .
- (vii) The k -ary *Ramsey quantifier* $\exists^{k\text{-ram}}$ is the class of structures $(A; E)$, $E \subset A^k$, for which there is an infinite $X \subset A$ such that for all pairwise distinct $x_1, \dots, x_k \in X$, $E(x_1, \dots, x_k)$.

The following general definition will allow us to compare the expressive strength of quantifiers.

Definition 1.4.3 Let Q be a quantifier, \mathbf{Q} a collection of quantifiers, and τ the signature of Q . Say that Q is *definable in* \mathbf{Q} if there is a sentence θ over the signature τ in the logic $\text{FO}[\mathbf{Q}]$ with $Q = \{\mathcal{A} \mid \mathcal{A} \models \theta\}$.

For instance, a structure $(A; X)$ satisfies $\exists^{(0,2)}z X(z) \vee \exists^{(1,2)}z X(z)$ if and only if X is finite. Hence \exists^∞ is definable in $\{\exists^{(0,2)}, \exists^{(1,2)}\}$.

Of course the generalised quantifiers that interest us most are the ones, like \forall and \exists , that preserve regularity.

Definition 1.4.4 Fix class \mathcal{C} as one of **S-AutStr**, **T-AutStr**, **ω S-AutStr**, or **T-AutStr**. Let Q be a quantifier with signature $\tau = (R_i)_{i \leq k}$, where R_i has associated arity r_i . Say that quantifier Q *preserves regularity for the class \mathcal{C}* if for every $n \in \mathbb{N}$, and every automatic presentation μ of a structure $\mathcal{A} \in \mathcal{C}$, every formula

$$Q\bar{x}_1, \dots, \bar{x}_k(\Psi_1^{\mathcal{A}}(\bar{x}_1, \bar{z}), \dots, \Psi_k^{\mathcal{A}}(\bar{x}_k, \bar{z}))$$

defines a relation R in \mathcal{A} with $\mu^{-1}(R)$ regular (here $\bar{z} = (z_1, \dots, z_n)$ and the Ψ_i are first-order \mathcal{A} -formulas).

Say that Q *preserves regularity effectively* if an automaton for $\mu^{-1}(R)$ can effectively be constructed from the automata of the presentation and the formulas Ψ_i .

Since not every structure is injectively presentable, we may restrict this definition to the class \mathcal{C} of injectively presentable structures from **ω S-AutStr** (or **ω T-AutStr**). For this, replace ‘automatic presentation’ with ‘injective automatic presentation’ in the above definition.

Example 1.4.5 The reachability quantifier is not regularity preserving (for any of the classes). For otherwise, by Example 1.3.13, the set of starting configurations that drive a given Turing Machine to a halting state would be regular, and hence computable.

The first steps have been taken in exploring those quantifiers that preserve regularity.

Theorem 1.4.6 Let \mathcal{C} be any of the following classes of structures **inj- ω T-AutStr**, **ω S-AutStr**, **T-AutStr**, **S-AutStr**.

- (i) The following unary quantifiers preserve regularity effectively for \mathcal{C} : $\exists^{\infty}, \exists^{mod}, \exists^{\leq\aleph_0}, \exists^{>\aleph_0}$ [20, 90, 94, 85, 9].
- (ii) Every unary quantifier that preserves regularity for the class **S-AutStr** is already definable from $\exists^{mod}, \exists^{\infty}$ [125].

The second item also implies that every unary quantifier that preserves regularity for the class **inj- ω S-AutStr** is already definable from $\exists^{mod}, \exists^{\infty}, \exists^{\leq\aleph_0}, \exists^{>\aleph_0}$. This is because for an ω -regular relation $R(\bar{x}, \bar{z})$ the cardinality of the set $R(-, \bar{c})$ (for any fixed parameter \bar{c}) is finite, countable or has size continuum [94].

Theorem 1.4.7 (see [125]) Each k -ary Ramsey quantifier preserves regularity effectively for the class **S-AutStr**.

Kuske and Lohrey observed that the proof of this theorem can be generalised to quantifiers of the form ‘there exists an infinite set X satisfying θ ’, where θ is a property of sets closed under taking subsets. They use this to show that certain problems, while Σ_1^1 -complete for recursive graphs, are decidable on automatic graphs [96].

Order-invariance

Definition 1.4.8 Fix a signature τ and a new symbol \leq . A formula $\phi(\bar{x})$ in the signature $\tau \cup \{\leq\}$ is called *order invariant* on a τ -structure \mathcal{A} if for all tuples \bar{a} from A and all linear orders \leq_1 and \leq_2 on A , we have that $(\mathcal{A}, \leq_1) \models \phi(\bar{a})$ if and only if $(\mathcal{A}, \leq_2) \models \phi(\bar{a})$. The *relation defined by the order invariant ϕ in \mathcal{A}* is the set of tuples \bar{a} from A such that $(\mathcal{A}, \leq) \models \phi(\bar{a})$ for some (and hence all) linear orders \leq on A .

The Fundamental Theorem can be extended on injective presentations to include order-invariant formulas in those cases where there is a regular linear ordering of the set $f^{-1}(A)$. On finite-words, finite-trees and ω -words there are regular linear orderings. However, we do not know if there is a regular linear ordering on the set of all ω -trees. On the other hand, certain separating examples from finite model theory are adaptable to the automatic world.

Proposition 1.4.9 ([5]) *There exists a structure $\mathfrak{B} \in \mathbf{S}\text{-AutStr}$ and an order-invariant definable relation S^* in \mathfrak{B} that is not definable in \mathfrak{B} using any extension of FO with only unary quantifiers.*

1.4.2 Complexity of some problems

First-order theories

By Theorem 1.3.4 query-evaluation and model-checking for first-order formulas are effective on automatic structures. However, the complexity of these problems is in general non-elementary, i.e. it exceeds any fixed number of iterations of the exponential function. For instance the first-order theories of the universal structures \mathcal{N}_k and $\mathcal{S}_{[k]}$ ($k \geq 2$) have non-elementary complexity [77] (cf. also the remark after Example 1.4.39).

There are various sensible ways of measuring model-checking complexity. First, one may fix a formula and ask how the complexity depends on the input structure. This measure is called *structure complexity*. On the other hand, *expression complexity* is defined relative to a fixed structure in terms of the length of the formula. Finally, one can look at the combined complexity where both parts may vary.

	Structure-Complexity ^a	Expression-Complexity
Model-Checking		
Σ_0	LOGSPACE-complete	ALOGTIME-complete ¹⁶
$\Sigma_0 + \text{func}$	NLOGSPACE	in quadratic time ¹⁶ and PTIME-complete
Σ_1	PTIME ¹⁷	PSPACE-complete (EXPTIME-c. for T-AutStr)
Σ_2	PSPACE-complete ¹⁷	EXPSPACE-complete (2EXPTIME-c. for T-AutStr)
Query-Evaluation		
Σ_0	LOGSPACE	PSPACE
Σ_1	PSPACE	EXPSPACE

Figure 1.3 Complexity of fragments of FO on automatic structures

^a Structure complexity is measured in terms of the size of the largest deterministic automaton in the input presentation.

In [25] Blumensath and Grädel studied the expression and structure complexity of model-checking and query evaluation for quantifier-free and existential first-order formulas both in a relational signature and allowing terms in quantifier-free formulas. Their results are complemented by those of Kuske and Lohrey [95] on the expression complexity of Σ_1 (existential) and Σ_2 formulas of a relational signature over arbitrary word- and tree-automatic structures. Figure 1.3 provides a summary.

On certain subclasses of automatic structures there is better complexity. In section 1.3.7 above we have mentioned that the first-order theory of each structure allowing a word-automatic presentation of polynomial density is decidable in PSPACE. Kuske and Lohrey [101, 95] studied automatic structures whose Gaifman graphs are of *bounded degree*. Relying on locality of first-order logic they have identified the expression complexity of FO model checking on word-automatic and tree-automatic structures of bounded degree to be 2EXPSPACE-complete and 3EXPTIME-complete, respectively. The combined complexity remains 2EXPSPACE for word-automatic presentations and is in 4EXPTIME for tree-automatic presentations. For finer results we refer to [95].

¹⁶ This is a generalisation of the quadratic solution of the word problem in automatic groups [31] (see Section 1.4.5).

¹⁷ Model checking with a fixed Σ_1 formula reduces to a membership or non-emptiness test for an NFA. For fixed Π_2 formulas the problem is polynomially equivalent to the universality problem of NFAs, and thus PSPACE-complete. (We thank Anthony To for pointing out the error in [25].)

Beyond first-order

A fundamental problem in verification is deciding *reachability*: whether there is a path between specified source and target nodes. Since the configuration space of an arbitrary Turing machine is finite-word automatic, the halting problem can be reduced to the reachability problem on the configuration graph of a universal Turing-machine. Similar reductions show the undecidability, over (finite-word) automatic structures, of connectivity, isomorphism, bisimulation and hamiltonicity [25, 96].

On the other hand there are natural classes of automatic structures for which these problems become decidable (see Figure 1.1). For instance, VRA-equational graphs have a decidable FO-theory with reachability and are finite-tree automatic. Reachability and connectivity in locally-finite unary-automatic graphs are in fact decidable in PTIME. Bisimulation equivalence of HR-equational graphs of finite out-degree is decidable [128] (see section 1.2.2).

Finally we mention some cases where full MSO is decidable. Prefix recognisable structures (which include the unary automatic structures) are finite-word automatic. A structure of the form $(\mathbb{N}, <, C_1, \dots, C_k)$ is called a colouring of the line. Every known finite-word automatic colouring of the line, and this includes every morphic sequence, has decidable MSO-theory (cf. Theorem 1.4.38 and see [7]). Furthermore, every word-automatic equivalence relation has a decidable MSO-theory. This follows from the above and the observation (Proposition 1.4.40) that if there are only finitely many infinite classes then the equivalence relation is FO-definable in some word-automatic colouring of the line [7].

Isomorphism problem

A measure of the complexity of a class of structures is the *isomorphism problem*, namely the problem of deciding, given two \square -automatic presentations \mathfrak{d} and \mathfrak{d}' , whether or not the structures they present are isomorphic.

The characterisations of the finite-word automatic Boolean algebras and ordinals [88, 63] imply that the isomorphism problem for each of these classes is decidable. Also, as noted, the isomorphism problem for equational graphs is decidable 1.2.10.

Configuration spaces of Turing machines are locally finite and the complexity of the isomorphism problem for locally-finite directed graphs in $\mathbf{S}\text{-AutStr}$ is Π_3^0 -complete [124]. However, by massaging the configuration spaces we get that the isomorphism problem for automatic graphs is as hard as possible: Σ_1^1 -complete. This is done by reducing the isomorphism

problem for computable structures, known to be Σ_1^1 -complete, to that of automatic structures.

Theorem 1.4.10 ([124]) *The complexity of the isomorphism problem for each of the following classes of $\mathbf{S}\text{-AutStr}$ structures is Σ_1^1 -complete:* (i) undirected graphs, (ii) directed graphs, (iii) successor trees, and (iv) lattices of height 4.

Problem 1.4.11 What is the exact complexity of the isomorphism problem for the following classes: ¹⁸

- (i) Automatic equivalence structures (easily seen to be Π_1^0).
- (ii) Automatic linear orders.

Traces

Infinite edge-labelled graphs, when viewed as infinite automata, can accept non-regular languages. Naturally, context-free graphs accept precisely the context-free languages. Though prefix-recognisable graphs form a structurally much richer class they have the same language accepting power as context-free graphs (cf. Theorem 1.2.11 items (1) and (6)). Graphs in the Caucal hierarchy have the same accepting power as higher-order pushdown automata (see Theorem 1.2.16) tracing languages on the corresponding levels of the OI-hierarchy of [62]. The traces of GTRS-graphs form a language class in between the context-free and context-sensitive classes of the Chomsky hierarchy [99]. Rational graphs accept precisely the context-sensitive languages [108]. All context-sensitive languages can in fact be accepted by word-automatic graphs [123], cf. also [35] for a more accessible proof and finer analysis. Meyer proved that the traces of tree-automatic graphs are those languages recognisable in ETIME, i.e. in $2^{\mathcal{O}(n)}$ time [103].

1.4.3 Non-automaticity via pumping and counting

It is usually quite simple to show that a structure has an automatic presentation (if indeed it does have one!). On the other hand, there are only a handful of elementary techniques for showing that a structure has no automatic presentation. Most rely on the pumping lemma of automata theory.

¹⁸ While this work has been in print, Kuske, Liu and Lohrey have greatly contributed to settling these and related questions. We refer to their forthcoming paper.

Sometimes we can provide a full characterisation of classes of automatic structures. The first non-trivial characterisation was for the word-automatic ordinals (in the signature of order).

Theorem 1.4.12 (Delhommé [63])

- (i) An ordinal α is in **S-AutStr** if, and only if, $\alpha < \omega^\omega$.
- (ii) An ordinal α is in **T-AutStr** if, and only if, $\alpha < \omega^{\omega^\omega}$.

A relation R is *(n + m) locally finite* if for every (x_1, \dots, x_n) there are only finitely many (y_1, \dots, y_m) such that $R(\bar{x}, \bar{y})$ holds. Obviously, every functional relation $f(\bar{x}) = y$ is locally finite. Other examples of locally finite relations are equal-length **e1**, length comparison $|y| < |x|$, and the prefix relation $y \prec_{\text{prefix}} x$. Note that local finiteness depends on the partitioning of the variables, e.g. $x \prec_{\text{prefix}} y$ is not locally finite.

A simple pumping argument gives the following important tool.

Proposition 1.4.13 (Elgot and Mezei [66]) *Let $R \subseteq (\Sigma^*)^{n+m}$ be a regular and locally finite relation. Then there is a constant k such that for all \bar{x}, \bar{y} satisfying R , $\max_j |y_j| \leq \max_i |x_i| + k$. In particular, if f is a regular function then there is a constant k such that for every \bar{x} in its domain we have $|f(\bar{x})| \leq \max_i |x_i| + k$.*

Growth of generations

Consider a structure \mathfrak{A} with functions $\mathcal{F} = \{f_1, \dots, f_s\}$ and a sequence $E = \{e_0, e_1, e_2, \dots\}$ of elements of \mathfrak{A} . The generations of E with respect to \mathcal{F} are defined recursively as follows.

$$\begin{aligned} G_{\mathcal{F}}^0(E) &= \{e_0\} \\ G_{\mathcal{F}}^{n+1}(E) &= G_{\mathcal{F}}^n(E) \cup \{e_{n+1}\} \\ &\quad \cup \{f(\bar{a}) \mid f \in \mathcal{F}, a_i \in G_{\mathcal{F}}^n(E) \text{ for each } i \leq |\bar{a}|\} \end{aligned}$$

We are interested in how fast $|G_{\mathcal{F}}^n(E)|$ grows as a function of n .

- Example 1.4.14** (i) Free semigroup on m generators: here $\mathcal{F} = \{\cdot\}$ and $E = \{e_1, \dots, e_m\}$. For $m \geq 2$, since $G_{\mathcal{F}}^m(E) \supset E$, the set $G_{\mathcal{F}}^{m+n}(E)$ includes all strings over E of length at most 2^n ; thus the cardinality of $G_{\mathcal{F}}^{m+n}(E)$ is at least a double exponential in n .
- (ii) If $p : D \times D \rightarrow D$ is injective then for $\mathcal{F} = \{p\}$ and $E = \{e_1, e_2\}$ (distinct elements of D) $|G_{\mathcal{F}}^n(E)|$ is at least a double exponential.

We now iterate Proposition 1.4.13.

Proposition 1.4.15 ([87],[20, 25]) *Let $\mathfrak{A} \in \mathbf{S}\text{-AutStr}$ and consider an injective presentation \mathfrak{d} with naming function f . Let \mathcal{F} be a finite set of functions FO-definable in \mathfrak{A} and $E = \{e_0, e_1, \dots\}$ a definable set of elements ordered according to length in \mathfrak{d} , i.e. $|f^{-1}(e_0)| \leq |f^{-1}(e_1)| \leq \dots$. Then there is a constant k such that for every n and for every $a \in G_{\mathcal{F}}^n$ $|f^{-1}(a)| \leq kn$. In particular, $|G_{\mathcal{F}}^n| = 2^{\mathcal{O}(n)}$.*

In other words, the number of elements that can be generated using functions is at most a single exponential in the number of iterations. Continuing the previous examples, neither the free semigroup nor any bijection $f : D \times D \rightarrow D$ (also called a pairing function) is word-automatic. It is trickier to apply the proposition to show that Skolem arithmetic (\mathbb{N}, \times) is not word-automatic (see [20, 25]). It is nevertheless tree-automatic, cf. Example 1.3.9.

The application of propositions 1.4.13 and 1.4.15 has been pushed to their limits:

- Proposition 1.4.16** (i) *If a group (G, \cdot) is word-automatic then every finitely generated subgroup is virtually Abelian (has an Abelian subgroup of finite index). In particular, a finitely generated group is in $\mathbf{S}\text{-AutStr}$ if, and only if, it is virtually Abelian [116, 114].*
- (ii) *A Boolean Algebra (in the signature $(\cup, \cap, \cdot^c, \perp, \top)$) is in $\mathbf{S}\text{-AutStr}$ if, and only if, it is a finite power of the Boolean Algebra of finite or co-finite subsets of \mathbb{N} [88]. In particular, the countable atomless Boolean Algebra is not in $\mathbf{S}\text{-AutStr}$.*
- (iii) *There is no infinite integral domain in $\mathbf{S}\text{-AutStr}$ [88].*
- (iv) *No word-automatic structure (D, R) has a subset $N \subset D$ such that (N, R) is isomorphic to (\mathbb{N}, \cdot) , cf. [114].*

The proof of the first item starts with the observation that every finitely-generated group $G \in \mathbf{S}\text{-AutStr}$ has polynomial density - that is, for every finite set $A = \{a_1, \dots, a_k\}$ the function

$$\gamma(n) = |\{\prod_{i < n} c_i^{\sigma_i} \mid \forall i < n : c_i \in A, \sigma_i \in \{1, -1\}\}|$$

is bounded by a polynomial (this exploits associativity of the group operation). The rest of the proof uses powerful theorems of Gromov and Ershov (see [114] for a survey of word-automatic groups).

Number of definable subsets

Various countable random structures, such as the random graph, do not have word- or tree-automatic presentations [88, 63]. The approach

to proving these facts has a model-theoretic flavour: for a purported automatic presentation, it involves counting the number of definable subsets of elements represented by words of bounded length.

Consider the usual definition of a set defined by φ with parameter b that remains fixed:

$$\varphi(-, b)^{\mathfrak{A}} = \{a \in \mathfrak{A} \mid \mathfrak{A} \models \varphi(a, b)\}.$$

A finite set $X \subset A$ is *fully shattered by φ* if the cardinality of the family

$$\{\varphi(-, b)^{\mathfrak{A}} \cap X \mid b \in A\}$$

is as large as possible, namely $2^{|X|}$. For instance, Benedikt et al. [16] observe that in $S_{[2]}$ each of the sets $\{0, 00, \dots, 0^n\}$ can be fully shattered by the formula $\varphi(x, b) = \exists z (\text{suc}_1 z \prec_{\text{prefix}} b \wedge \text{el}(z, x))$.

By contrast, in every automatic presentation with naming function f and domain $D \subseteq \Sigma^*$, the image under f of each $D_{\leq n} := D \cap \Sigma^{\leq n}$ can only be linearly shattered by definable families.

Proposition 1.4.17 ([88, 63]) *In every automatic presentation of a structure \mathfrak{A} with naming function f and for every formula φ :*

$$|\{\varphi(-, b)^{\mathfrak{A}} \cap f(D_{\leq n}) \mid b \in A\}| = \mathcal{O}(|f(D_{\leq n})|).$$

As an application recall that the random graph is characterised by the property that for every partition of a finite set X of vertices into sets U and V , there is a vertex b connected to all elements of U and to no element of V . In other words, every finite set X of vertices is fully shattered by the edge relation as the parameter b is varied. So by Proposition 1.4.17 the random graph has no word-automatic presentation. Similar reasoning yields the following.

Proposition 1.4.18 ([88, 63]) *The following are not in $\mathbf{S}\text{-AutStr}$: the random graph, the random partial order, the random K_n -free graph.*

Using Theorem 1.3.25 one can establish non-automaticity of the random graph in a far more general sense.

Theorem 1.4.19 ([47]) *Neither the random graph nor the free monoid on two generators is finite-tree automatic with any oracle.*

In fact neither is ω -word automatic with any oracle, as witnessed by the following theorem which follows from the proof of Theorem 1.3.31.

Theorem 1.4.20 *If a countable structure is ω -word automatic with oracle, then it is also finite-word automatic with (the same) oracle.*

1.4.4 Comparing presentations

When we think of an automatic structure we frequently have a particular automatic presentation in mind. Some structures have *canonical* presentations. For instance, (a^*, \leq_{len}) is arguably the canonical presentation of $(\mathbb{N}, <)$ and $(\{0, 1\}^*, \mathbf{suc}_0, \mathbf{suc}_1, \prec_{\text{prefix}}, \mathbf{e1})$ is the canonical presentation of itself. Some well-known structures have *natural* presentations, none of which can be indisputably called canonical. The base $k \in \mathbb{N}$ ($k > 1$) presentations of $(\mathbb{N}, +)$ can be considered equally natural; but then what about the Fibonacci numeration system? The field of *regular numeration systems*, though using a somewhat different terminology, investigates automatic presentations of $(\mathbb{N}, +)$ and ω -word automatic presentations of $(\mathbb{R}, +)$. Finally, there are *pathological* presentations that are used to pin down the relationship between definability in a structure and regularity in its presentations [90].

How are we to compare different automatic presentations of the same structure? What are the crucial aspects of a presentation that distinguish it from others?

Canonical representations of context-free graphs were investigated by Séniergues. In [127] a p-structure for a graph G is a PDA \mathcal{A} (having no ϵ -transitions) together with an isomorphism between the configuration graph of \mathcal{A} and G . Furthermore, a p-structure for G is *P-canonical* if the distance in G between a vertex v and the root is equal to the stack height of the configuration representing v (cf. [112]'s notion of a canonical automaton for a context-free graph; and [41, 44]). For a fixed graph G Séniergues considers two p-structures equivalent if there is a rational isomorphism between them, and shows that every equivalence class of p-structures contains a P-canonical one [127].

An example from the theory of numeration systems is provided by the celebrated result of Cobham and Semenov. Recall that naturals p and q are called *multiplicatively independent* if they have no common power (ie. $p^k \neq q^l$ for all $k, l \geq 1$) and *multiplicatively dependent* otherwise.

Theorem 1.4.21 (Cobham-Semenov¹⁹, cf. [26, 19, 109])

The following dichotomy holds for $p, q \geq 2$.

- (i) If p and q are multiplicatively dependent then a relation $R \subseteq \mathbb{N}^r$ is regular when coded in base p iff it is regular when coded in base q .
- (ii) If p and q are multiplicatively independent then a relation $R \subseteq \mathbb{N}^r$ is regular in both base p and base q iff R is FO-definable in $(\mathbb{N}, +)$.

¹⁹ Cobham proved it for sets; Semenov later extended it to arbitrary relations.

The meaning of (i) is that, for instance, bases 2^l and 2^k are expressively equivalent. There is a very simple coding translating numerals between these bases, which bijectively maps blocks of k digits in the first system to blocks of l digits in the second system. Every pair of multiplicatively dependent numeration systems are linked by similar translations.

According to (ii) the base 2^k presentation is as different as it can be from, say, the base 3 presentation. This point is further stressed by the following result of Bés based on the work of Michaux and Villemaire.

Theorem 1.4.22 ([18]) *Let p and q be multiplicatively independent, and $R \subseteq \mathbb{N}^r$ regular when coded in base q , but not first-order definable in $(\mathbb{N}, +)$. Then the first-order theory of $(\mathbb{N}, +, |_p, R)$ is undecidable.*

On a similar note we introduce the following general notions.

Definition 1.4.23 (Subsumption and equivalence)

Consider two \square -automatic presentations of some structure \mathfrak{A} with naming functions f and g , respectively. We say that f *subsumes* g ($g \preceq f$) if for every relation R over the domain of \mathfrak{A} , if $g^{-1}(R)$ is \square -regular then $f^{-1}(R)$ is \square -regular. If both $f \preceq g$ and $g \preceq f$ then we say that the two presentations are *equivalent* and write $f \sim g$. Moreover, we say that a \square -automatic presentation of \mathfrak{A} is *prime* if it is subsumed by all other \square -automatic presentations of \mathfrak{A} .

word-automatic presentations

The definition of equivalence of automatic presentations is modelled on case (i) of Theorem 1.4.21. In [5] it has been shown that two finite-word automatic presentations are equivalent if and only if the transduction translating names of elements from one presentation to the other is computable by a *semi-synchronous transducer*: a two-tape finite automaton processing its first tape in blocks of k letters and its second tape in blocks of l letters for some fixed positive k and l . (Note that, except in trivial cases, k/l is uniquely determined [5].)

Theorem 1.4.24 ([5]) *Two finite-word automatic presentations of some $\mathfrak{A} \in \mathbf{S}\text{-AutStr}$ with naming functions $f_i : D_i \rightarrow A$, $i \in \{1, 2\}$, are equivalent if, and only if, the transduction $T = \{(x, y) \in D_1 \times D_2 \mid f_1(x) = f_2(y)\}$ translating names of elements from one presentation to the other is semi-synchronous rational.*

Corollary 1.4.25 *Let f_1 and f_2 be naming functions of equivalent automatic presentations of \mathfrak{A} . Then there is a constant C such that*

for every n -ary relation R over $\text{dom}(\mathfrak{A})$ and for every automaton \mathcal{A}_1 recognising $f_1^{-1}(R)$ there is an automaton \mathcal{A}_2 of size $|\mathcal{A}_2| \leq C^n \cdot |\mathcal{A}_1|$ recognising $f_2^{-1}(R)$, and vice versa.

Let \mathfrak{U} be one of the universal finite-word automatic structures \mathcal{S}_Σ (for $|\Sigma| > 1$), $\mathcal{P}_f(\Delta_1)$, or $(\mathbb{N}, +, |_k)$ (for $k > 1$). Using semi-synchronous translations one can establish the following.

Theorem 1.4.26 ([5, 6]) *The universal structure \mathfrak{U} has only a single word-automatic presentation up to equivalence.*

The assertion of the theorem can be reformulated as follows.

Corollary 1.4.27 *For a relation R , the expansion (\mathfrak{U}, R) is in $\mathbf{S}\text{-AutStr}$ if, and only if, R is FO-definable in \mathfrak{U} .*

The prime presentation of a structure, if one exists, is unique up to equivalence, hence may as well be called canonical. The unary presentation of $(\mathbb{N}, <)$ is a prime word-automatic presentation. It is, however, not a prime presentation of (\mathbb{N}, suc) , which allows, for every $m > 1$ a word-automatic presentation in which divisibility by m is not regular [90]. It can be inferred that (\mathbb{N}, suc) has no prime presentation.

Recall Theorem 1.3.27 stating that each word-automatic presentation, of structure \mathfrak{A} , over a domain of polynomial density of degree d directly corresponds to a d -dimensional interpretation of \mathfrak{A} in the structure $\mathfrak{M} = (\mathbb{N}, <, \{\equiv_{(\text{mod } m)}\}_{m>1})$, and hence also in $(\mathbb{N}, +)$. So every p-automatic structure has infinitely many pairwise incomparable word-automatic presentations ‘inherited’ from $(\mathbb{N}, +)$, namely, based on different numeration systems.

In fact, \mathfrak{M} allows a non-trivial 2-dimensional interpretation in itself. Simply consider the lexicographic ordering of all pairs (n_1, n_2) such that $n_1 \geq n_2$ as an interpretation of $(\mathbb{N}, <)$ and observe that moduli of positions within the lexicographic ordering of tuples can be expressed in terms of moduli of their components. Thus, by composing interpretations, every p-automatic presentation of \mathfrak{M} is properly subsumed by other p-automatic presentations with domains of asymptotically greater polynomial densities. This carries over to all p-automatic structures.

In contrast, from results of [5, 8] it follows that $g \preccurlyeq f$ implies $g \sim f$ for any two word-automatic presentations of a given structure, provided that either both f and g have domains of exponential density, or both have a domain of polynomial density of the same degree.

Therefore, the height of the partial order of word-automatic presen-

tations of \mathfrak{A} under subsumption and modulo equivalence is ω if \mathfrak{A} is p-automatic and 1 if \mathfrak{A} is not p-automatic. It is not known whether the width of the subsumption order modulo equivalence is always one or infinite for word-automatic structures that are not p-automatic.

tree-automatic presentations

Colcombet and Löding [47] investigated the power of finite-subset interpretations applied to arbitrary trees. In our terminology these are tree-automatic presentations with arbitrary oracles.

In the tree-automatic model the analogue of Theorem 1.4.26 does not hold. A tree-automatic presentation of $\mathcal{P}_f(\Delta_2)$ incomparable with the natural one can be forged simply by ‘folding each tree in half about the vertical axis’, i.e. taking the mirror image of the subtree below the right child of the root and smoothly combing it together with the untouched left half, e.g. as in Example 1.3.11(3). Despite this, the fact concerning primality of the natural presentation of the universal structure holds in an even stronger sense.

Proposition 1.4.28 ([47, Lemma 5.6]) *The natural tree-automatic presentation with oracle O and with the identity naming function of the finite-subset envelope $\mathcal{P}_f(\mathfrak{T}_O)$ of the oracle tree \mathfrak{T}_O is a prime presentation with respect to tree-automatic presentations with arbitrary oracle.*

In particular, ‘the’ word-automatic presentation of $\mathcal{P}_f(\Delta_1)$ and the natural tree-automatic presentation of $\mathcal{P}_f(\Delta_2)$ are both prime even among tree-automatic presentations with arbitrary oracles. This is complemented by the following result of [47].

Theorem 1.4.29 *All tree-automatic presentations of $\mathcal{P}_f(\Delta_1)$ are equivalent.*

Therefore, the same holds true for all of the universal structures from Theorem 1.4.26.

1.4.5 Other notions of automaticity

Specific automatic presentations have been employed in other mathematical fields: computational group theory [31], symbolic dynamics [13], numeration systems (of integers or reals) [76], and infinite sequences represented in natural numeration systems [2, 26, 4]. In this section we survey natural presentations of certain structures that have mostly been considered independently of the general theory of automatic structures.

Automatic groups

Thurston (1986) motivated by work of Cannon on hyperbolic groups introduced the notion of automatic groups. A finitely generated group G is *automatic* in this sense if for some set of semigroup generators S and associated canonical homomorphism $f : S^* \rightarrow G$

- (i) there is a regular language $W \subset S^*$ so that f restricted to W is surjective,
- (ii) for every s a generator from S or the group identity, the following binary relation over W is regular:

$$\{(u, v) \mid f(u) = f(v)s\}.$$

This is in fact an algebraic notion: it does not depend on the particular choice of generators. From the automata presenting the group one can extract a finite presentation of the group, and a quadratic-time algorithm deciding the word problem.

Proposition 1.4.30 (k -fellow traveler property) *A group G with semigroup generators $S = \{s_1, \dots, s_r\}$ is automatic if, and only if, there exists a regular set $W \subseteq S^*$ and $k \in \mathbb{N}$ such that $f|_W$ is surjective and W satisfies the k -fellow traveler property:*

$\forall u, v \in W \text{ with } d(u, v) \leq 1 \forall i \leq \max\{|u|, |v|\} : d(u_1 \dots u_i, v_1 \dots v_i) \leq k$

where $d(u, v)$ denotes the length of the shortest path between u and v in the Cayley graph of G with generators S .

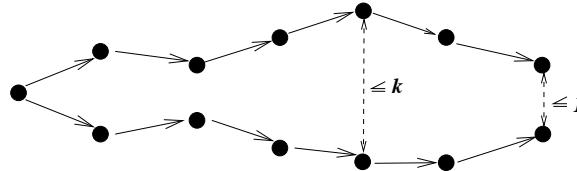


Figure 1.4 k -fellow traveler property.

Virtually Abelian groups and Gromov's word hyperbolic groups constitute important examples of automatic groups in this sense. Major results of this programme are presented in [31] (see also the introductions by Farb [71] and by Choffrut [46]).

More recently, this notion has been extended to semigroups [29, 30, 84, 28] and monoids [83, 129, 102].

Let us compare the following three notions: (i) groups whose multiplication function admits a word-automatic presentation, (ii) finitely generated automatic groups, and (iii) finitely generated groups with a Cayley graph admitting a word-automatic presentation. It is known [116] that a finitely generated group allows a word-automatic presentation of type (i) iff it is virtually Abelian. All virtually Abelian finitely generated groups are automatic in the sense of this subsection. Hence (i) implies (ii) for finitely generated groups. Furthermore, by definition, the Cayley graph of every automatic group has a word-automatic presentation. Hence (ii) implies (iii), but the converse fails. As Séniérgues has pointed out the Heisenberg group is not automatic even though its Cayley graph has an automatic presentation. For further reading we recommend the survey by Nies [114].

Generalised numeration systems

The theory of *generalised numeration systems* [76] is concerned with representations of \mathbb{N} and \mathbb{R} in various bases and using different (possibly negative) digits. In general, the basis $U_0 < U_1 < U_2 < \dots$ of the system does not have to be the sequence of powers of a natural. One considers bases satisfying appropriate linear recursions, or alternatively powers of a base β which is the greatest root of a polynomial of a certain type. The study of generalised numeration systems goes back to Rényi who in 1957 introduced β -expansions.

Without going into the particulars of this very rich field we point out that a number may have more than one representation in a given numeration system. Thus from a practical perspective one is interested in *normalised* numerals obtained via the *greedy* algorithm. Normalised numerals are ordered according to $<_{\text{lex}}$ (length and then lexicographically, most significant digit first). A regular set of (normalised) numerals $N \subseteq [d]^*$ over the set of digits $0, \dots, d - 1$ is simply an automatic copy of $(\mathbb{N}, <)$ of the form $(N, <_{\text{lex}})$.

A fundamental question in this context asks under which circumstances addition can be computed by a synchronous finite automaton. When this is the case one speaks of a *regular numeration system*. On this matter we refer to [76] and the references therein.

Example 1.4.31 The *Fibonacci numeration system* is a prominent example of a regular numeration system. It has the Fibonacci numbers $1, 2, 3, 5, 8, \dots$ as its basis, and the binary digit set. The normalised numerals delivered by the greedy algorithm are $\varepsilon, 1, 10, 100, 101, 1000,$

1001, 1010, 10000, 10001, ... in the length-lexicographic ordering. They are the binary strings avoiding 11 as a factor since greedy normalisation prefers 100 to 11. Naturally, 10^n represents the n th Fibonacci number.

More generally we ask how can one classify the word-automatic presentations of $(\mathbb{N}, +)$? Or those of $(\mathbb{N}, <)$? Below we survey known classes of automatic presentations of expansions of $(\mathbb{N}, <)$ by unary predicates, i.e. infinite sequences.

Automatic sequences

The theory of *automatic sequences* [2] studies ω -words representable in more-or-less standard numeration systems. Presentations of primary concern are those of base $k \in \mathbb{N}$, or of base $-k$, and possibly involving negative digits.

Definition 1.4.32 A sequence $s : \mathbb{N} \rightarrow \Sigma$ is *k-automatic* if for every $a \in \Sigma$ the set N_a of numerals in the standard base k numeration system representing all positions n such that $s(n) = a$ constitutes a regular language.

These *k-automatic sequences* have been characterised in both algebraic and logical terms. In order to formulate another characterisation some notions are required. A morphism $\varphi : \Gamma^* \rightarrow \Sigma^*$ is said to be *k-uniform* if $|\varphi(a)| = k$ for each $a \in \Gamma$. *Codings* are 1-uniform morphisms. A morphism $\varphi : \Gamma^* \rightarrow \Gamma^*$ is *prolongable* on some $a \in \Gamma$ if a is the first symbol of $\varphi(a)$. In this case the sequence $(\varphi^n(a))_{n \in \mathbb{N}}$ converges to either a finite or infinite word, which is a fixed point of φ , denoted $\varphi^\omega(a)$.

Theorem 1.4.33 ([26, 2]) *For any sequence $s : \mathbb{N} \rightarrow \Sigma$ the following are equivalent:*

- (1) *s is k-automatic;*
- (2) *the k-kernel of s: $\{(s_{nk^m+r})_n \mid r, m \in \mathbb{N}, r < k^m\}$ is finite;*
- (3) *the sets $s^{-1}(a)$ are FO-definable in $(\mathbb{N}, +, |_k)$ for each $a \in \Sigma$;*
- (4) *$s = \sigma(\tau^\omega(a))$ for some k-uniform morphism τ on some Γ^* and a coding $\sigma : \Gamma \rightarrow \Sigma$;*
- (5) *(assuming k is a prime and $\Sigma \subseteq \{0, \dots, k-1\}$): the formal power series $S(x) = \sum_n s_n x^n \in \mathbb{F}_k[[x]]$ is algebraic over $\mathbb{F}_k[x]$.*

For example, consider the morphism $\tau : 0 \mapsto 01, 1 \mapsto 10$. Its fixed point $\tau^\omega(0)$ is the *Thue-Morse sequence* $t = 01101001100101101001\dots$. This is a truly remarkable sequence bearing a number of characterisations and combinatorial properties [3]. For instance, its n th digit is 1 if, and only

if, the binary numeral of n contains an odd number of 1's. The 2-kernel of t is $\{t, \bar{t}\}$, where \bar{t} is obtained from t by flipping every bit.

Morphic words

One obtains a definition of *morphic words* by relaxing characterisation (4) of the above theorem. Morphic words thus constitute a generalisation of automatic sequences. They and their relatives have been extensively studied in the context of formal language theory, Lindenmayer systems and combinatorics on words.

Definition 1.4.34 *Morphic words* are those of the form $\sigma(\tau^\omega(a))$ for arbitrary homomorphism τ prolongable on a and arbitrary homomorphism $\sigma : \Gamma^* \rightarrow \Sigma^*$ extended to ω -words in the obvious way.

Example 1.4.35 Consider $\tau : a \mapsto ab, b \mapsto ccb, c \mapsto c$ and $\sigma : a, b \mapsto 1, c \mapsto 0$ both homomorphically extended to $\{a, b, c\}^*$. The fixed point of τ starting with a is the word $abccbcccbc^6b\dots$, and its image under σ , $110010^410^610^81\dots$, is the characteristic sequence of the set of squares. In general, for every strictly positive \mathbb{N} -rational sequence (s_k) the characteristic sequence of the set $\{\sum_{k=0}^n s_k \mid n \in \mathbb{N}\}$ is morphic [38]. This result also follows from Proposition 1.4.37.

While k -automatic sequences allow automatic presentations over the set of standard base k numerals, the above example suggests that morphic words may need generalised numeration systems. Indeed, every morphic word is automatically presentable in the following sense.

Consider a finite ordered alphabet $\Gamma = \{a_1 < a_2 < \dots < a_r\}$. In the induced *length-lexicographic order*, denoted $<_{\text{lex}}$, words over Γ are ordered according to their length first, while words of the same length are ordered lexicographically. Thus $(D, <_{\text{lex}})$ provides an automatic presentation of $(\mathbb{N}, <)$ for every infinite regular language D over Γ . Base k as well as so called generalised numeration systems are special cases of this scheme. The following notion thus generalises Definition 1.4.32.

Definition 1.4.36 We say that an ω -word $w : \mathbb{N} \rightarrow \Sigma$ is *length-lexicographically presentable* if there is an automatic presentation $(D, <_{\text{lex}})$ of $(\mathbb{N}, <)$ with naming function $f : D \rightarrow \mathbb{N}$ such that the sets $f^{-1}(w^{-1}(a))$ are regular for each $a \in \Sigma$.

It is not hard to see that an ω -word is length-lexicographically presentable if and only if it is morphic. There is a perfectly natural correspondence between the morphisms generating a word and the automaton

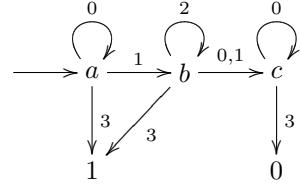
recognising the set of ‘numerals’, which, when length-lexicographically ordered, give an automatic presentation of the morphic word.

Proposition 1.4.37 ([122]) *An ω -word w is length-lexicographically presentable if, and only if, w is morphic.*

We illustrate the transformation from one formalism to the other on the characteristic sequence of squares from Example 1.4.35. Recall that it is generated by the following morphism τ and final substitution σ

$$\begin{aligned}\tau : \quad a &\mapsto ab & b &\mapsto ccb & c &\mapsto c \\ \sigma : \quad a &\mapsto 1 & b &\mapsto 1 & c &\mapsto 0\end{aligned}$$

The idea is to interpret symbols $\{a, b, c, 0, 1\}$ as states. Without loss of generality, the alphabets of the ranges of σ and τ are disjoint. The alphabet Γ of the automatic presentation consists of digits ranging from 0 to $|\tau| + |\sigma| - 1$, where $|\tau|$ is the maximum of $|\tau(x)|$ with $x \in \{a, b, c\}$ and $|\sigma|$ is defined similarly. Letters of the alphabet, ordered as usual, are used to index positions within the right-hand side of a τ -rule, or, when larger, positions inside the right-hand side of a substitution via σ .



The domain D of the presentation is recognised by the above automaton with both 1 and 0 as final states. With only 1 as a terminal state, the automaton recognises the numerals representing a square relative to the length-lexicographic enumeration of D . Starting with a deterministic automaton this transformation can be reversed producing a morphism τ representing the transition function linearised according to the ordering on the alphabet and with σ identified by the terminal states.

The MSO-theory of the structure $(\mathbb{N}, <, (w^{-1}(a))_a)$ for morphic w is decidable [38]. Moreover, the class of morphic words is closed under MSO-definable recolourings, i.e. under *deterministic generalised sequential mappings* [118]. These results are generalised by the following one, which can be seen as an extension of the Fundamental Theorem 1.3.4.

Theorem 1.4.38 ([7]) *Let $\mathfrak{d} = (D, <_{\text{llex}}, \bar{P})$ be a length-lexicographic presentation of a morphic word w and let $\varphi(\bar{x})$ be an $\text{MSO}[<, \bar{P}]$ -formula*

having only first-order variables free. Then there is an automaton \mathcal{A} , computable from \mathfrak{d} and φ and such that $(\mathfrak{d}, \mathcal{A})$ is a word-automatic presentation of w expanded by the relation defined by φ .

Caucal has shown that morphic sequences can be constructed as graphs on the second level of the pushdown hierarchy (cf. Definition 1.2.14) [43]. However, there are automatically presentable ω -words on higher levels as well.

Higher-order morphic words

Higher-order morphic words were introduced in [4, 7]. Morphic words of order k can be defined either in the style of Definition 1.4.34 based on a notion of ‘morphisms of order- k stacks’ or similar rules, or as in Definition 1.4.36 as those having an automatic presentation using the ‘ k -fold nested length-lexicographic order’ induced by an ordered alphabet. Theorem 1.4.38 extends to these automatic presentations of higher-order morphic words. The classes of order k morphic words form an infinite hierarchy, and are constructible on the $2k$ -th level of the pushdown hierarchy [7].

Example 1.4.39 As an example we mention the Champernowne word (cf. Example 1.3.23) obtained by concatenating decimal numerals in their usual order:

$$C = 1234567891011121314\dots$$

It is on the second level of this hierarchy (and on the fourth level of the pushdown hierarchy). Consider the level 2 morphism Δ given by the following intuitive production rules

$$\begin{aligned} S_x &\rightarrow S_x A_{\tau_1(x)} \dots A_{\tau_9(x)} \\ A_x &\rightarrow A_{\tau_0(x)} A_{\tau_1(x)} \dots A_{\tau_9(x)} \end{aligned}$$

where each τ_i is a (level 1) morphism of words in the usual sense mapping each digit $d \in \{0, \dots, 9\}$ to d and $\#$ to $i\#$. Applying Δ repeatedly to the initial level 2 stack $S_\#$ yields the following converging sequence

$$\begin{aligned} S_\# &\rightarrow S_\# A_{1\#} A_{2\#} \dots A_{9\#} \\ &\rightarrow S_\# A_{1\#} A_{2\#} \dots A_{9\#} A_{10\#} \dots A_{19\#} \dots \dots A_{90\#} \dots A_{99\#} \\ &\rightarrow \dots \end{aligned}$$

Hence C can be specified as $C = \sigma(\Delta^\omega(S_\#))$ with the morphism σ erasing all $\#$'s while preserving the other (level 1) symbols.

To give a word-automatic presentation we take the domain D to be

comprised of all words of the form $d_1m_1d_2m_2\dots d_sm_s$ with $d_1d_2\dots d_s$ a conventional decimal numeral and $m_1m_2\dots m_s = \text{o}^i\text{x}\text{o}^{s-i-1}$ a marker indexing the i th digit of this numeral. Elements of the domain are ordered using the length-lexicographic ordering in a nested fashion: comparing numerals (i.e. odd positions) first, and then according to the position of the marker x.

The Champernowne word contains every finite word over $\{0, 1, \dots, 9\}$ as a factor. The satisfiability problem of first-order logic on finite words, known to be non-elementary [79], is thus expressible in the FO theory of the Champernowne word, which is therefore also non-elementary. For the same reason the Champernowne word is not morphic. Every morphic word is MSO-definable in the Champernowne word, and every word-automatic equivalence structure having only finitely many infinite equivalence classes is interpretable in a second-order morphic word [7].

Proposition 1.4.40 *Consider $\mathfrak{A} = (A, E)$ with E an equivalence relation having, for each $n > 0$, $f(n) \in \mathbb{N}$ many equivalence classes of size n , and no infinite classes. Then $\mathfrak{A} \in \mathbf{S}\text{-AutStr}$ if, and only if, there is a second-order morphic word $w = 0^{m_0}10^{m_1}10^{m_2}1\dots$ such that $f(n) = |\{i \mid m_i = n\}|$.*

It remains open whether the decidability and definability results for MSO hold for all word-automatic infinite sequences. We are intrigued whether the isomorphism problem of automatic ω -words, or more broadly for automatic scattered linear orders, is decidable. Already for morphic words this is a notorious long-standing open problem.

1.5 Automatic Model Theory

We may reformulate the original problem — we seek a class of finitely-presentable structures \mathcal{C} that has an interesting model theory and lies somewhere between the finite structures (finite model theory) and all structures (classical model theory).

The richest and oldest class consists of the computable structures — these are structures whose domain and atomic relations are computable by Turing machines [70]. In computable model theory, a common theme is to take classical results from mathematics and model theory and to see to what extent they can be made effective. Here are two illustrative observations:

- (i) A computable (consistent) first-order theory has a computable model. Indeed, Henkin's construction can be seen as an algorithm computing the domain and atomic relations.
- (ii) Every two computable presentations of the rational ordering $(\mathbb{Q}, <)$ are computably isomorphic. Again, the standard back-and-forth argument can be seen as an algorithm building the isomorphism.

The program of feasible mathematics in the 1980's included the development of polynomial-time model theory [45]. However, every relational computable structure is isomorphic (in fact computably isomorphic) to a polynomial-time structure. Automatic structures can be seen as a further restriction of this class, and in fact this is the motivation in [87]. In this section we discuss some aspects of the model theory of automatic structures, a subject still in its infancy.

We split our discussion along two lines: model theory of the class **S-AutStr**, and model theory of the particular universal structure $\mathcal{S}_{[2]}$ (cf. Theorem 1.3.17).

1.5.1 Model theory restricted to the class of word-automatic structures

Blumensath shows that, as expected, certain notions of model theory fail when restricted to the class of automatic structures.

- Proposition 1.5.1** (i) *It is undecidable whether an FO-formula has a word-automatic model.*
- (ii) *The following properties fail on the class of word automatic structures: compactness, Beth, Interpolation, and Los-Tarski.*

The proofs are based on the observation that there is a FO formula which has automatic models of every finite cardinality but no infinite automatic models.

Löwenheim-Skolem

An automatic version of the Downward Löwenheim-Skolem Theorem would say that every uncountable ω -automatic structure has a countable elementary substructure that is also ω -automatic. Unfortunately this is false since there is a first-order theory with an ω -automatic model but no countable ω -automatic model. Indeed, consider the first-order theory of atomless Boolean Algebras. Kuske and Lohrey [94] have observed that it

has an uncountable ω -automatic model (namely the algebra from Example 1.3.12.4). However, Khoussainov et al. [88] show that the countable atomless Boolean algebra is not automatic and so, by Theorem 1.4.20, not ω -automatic either.

Here is the closest we can get to an automatic Downward Löwenheim-Skolem Theorem for ω -automatic structures.

Proposition 1.5.2 ([85]) *Let $(D, \approx, \{R_i\}_{i \leq \omega})$ be an omega-automatic presentation of \mathfrak{A} and let \mathfrak{A}_{up} be its restriction to the ultimately periodic words of D . Then \mathfrak{A}_{up} is a countable elementary substructure of \mathfrak{A} .*

Proof Relying on the Tarski-Vaught criterion for elementary substructures we only need to show that for all first-order formulas $\varphi(\bar{x}, y)$ and elements \bar{b} of \mathfrak{A}_{up}

$$\mathfrak{A} \models \exists y \varphi(\bar{b}, y) \Rightarrow \mathfrak{A}_{up} \models \exists y \varphi(\bar{b}, y).$$

By Theorem 1.3.4 $\varphi(\bar{x}, y)$ defines an omega-regular relation and, similarly, since the parameters \bar{b} are all ultimately periodic the set defined by $\varphi(\bar{b}, y)$ is omega-regular. Therefore, if it is non-empty, then it also contains an ultimately periodic word, which is precisely what we needed. \triangleleft

An identical proposition, also independently noted by Khoussainov and Nies, holds for $\mathfrak{A} \in \omega\text{T-AutStr}$ with regular trees in place of ultimately periodic words.

Consider the natural, say, binary ω -automatic presentation of $(\mathbb{R}, +)$. Its restriction to the set of elements represented by ultimately periodic ω -words is isomorphic to the additive group of the rationals $(\mathbb{Q}, +)$. Tsankov [136] has shown that there is no automatic divisible torsion-free Abelian group (DTAG). Hence the theory of DTAGs is another example of a first-order theory having an uncountable ω -automatic model but no countable (ω -)automatic models.

Automatic theorems König's Lemma

König's Lemma says that an infinite finitely-branching tree has an infinite path. We split our discussion of automatic analogues along two lines, depending on whether the signature is that of partial order (T, \preceq) or successor (T, S) .

Theorem 1.5.3 ([91]) *If $\mathcal{T} = (T, \preceq)$ is an automatic copy of an infinite finitely-branching tree, then \mathcal{T} has a regular infinite path. That is, there exists a regular set $P \subseteq T$ where P is an infinite path of \mathcal{T} .*

Proof Define a set P as those elements x such that $\exists^\infty w[x \prec w]$ and for which every $y \prec x$ satisfies that

$$\forall z, z' \in S(y)[z \preceq x \Rightarrow z \leq_{lex} z'].$$

Then P is the length-lexicographically least infinite path of \mathcal{T} (in the ordering induced by the finite strings presenting the tree). \triangleleft

However, using the 2-Ramsey quantifier we can do more.

Theorem 1.5.4 ([91]) *If $\mathcal{T} = (T, \preceq)$ is an automatic copy of a tree with countably many infinite paths, then every infinite path is regular.*

Proof Denote by $E(\mathcal{T}) \subseteq T$ the set of elements of a tree \mathcal{T} that are on infinite paths. It is definable in \mathcal{T} using the 2-Ramsey quantifier, so Theorem 1.4.7 gives that $E(\mathcal{T})$ is regular. Then every isolated path of \mathcal{T} is regular, since it is definable as $\{x \in E(\mathcal{T}) \mid p \preceq x\} \cup \{x \in E(\mathcal{T}) \mid x \prec p\}$, for suitable $p \in E(\mathcal{T})$. Replace \mathcal{T} by its derivative $d(\mathcal{T})$, which is also automatically presentable. Since the CB-rank of \mathcal{T} is finite [91] and $d^{\text{CB}}(\mathcal{T})(\mathcal{T})$ is the empty tree, every infinite path is defined in this way. \triangleleft

However, automatic successor trees behave more like computable trees:

Theorem 1.5.5 ([96]) *The problem of deciding, given automata presenting a successor tree (T, S) , whether or not it has an infinite path, is Σ_1^1 -complete.*

The proof consists of a reduction from the problem of whether a non-deterministic Turing machine visits a designated state infinitely often.

We compare with the computable case.²⁰ Fix the computable presentation of the full binary tree as consisting of the finite binary sequences with the immediate successor relation (so in fact the prefix relation is also computable). To stress this presentation, we refer to the tree as 2^ω . Similarly fix a natural computable presentation ω^ω of the ω -branching tree. A *computable subtree* of either of these trees is a computable prefix-closed subset.

- (i) There is an infinite computable subtree of 2^ω with no computable infinite path.
- (ii) There is a computable subtree of ω^ω with exactly one infinite path, and this path is not computable.
- (iii) The set of indices of computable subtrees of the binary tree 2^ω with at least one infinite path is Π_2^0 -complete.

²⁰ Thanks to Frank Stephan for discussions concerning this case.

- (iv) The set of indices of computable subtrees of ω^ω with at least one infinite path Σ_1^1 -complete.

Cantor's Theorems

One of Cantor's theorems says that every countable linear ordering embeds in the rational ordering \mathbb{Q} . The standard proof is easily seen to be effective given a computable presentation of $(\mathbb{Q}, <)$.

There are potentially a variety of automatic versions. The following proposition is the best known.

Proposition 1.5.6 [93] *Every automatic copy \mathcal{M} of a linear order can be embedded into some automatic copy of \mathbb{Q} by a function $f : \mathcal{M} \rightarrow \mathbb{Q}$ with the following properties:*

- (i) *The function f is continuous with respect to the order topology.*
- (ii) *The graph of f is regular.*

It is not known whether there is a single automatic copy of \mathbb{Q} that embeds, in the sense above, all automatic copies of all automatically presentable linear orders \mathcal{M} .

Cantor also proved that \mathbb{Q} is homogeneous: For every two tuples $x_1 < \dots < x_m$ and $y_1 < \dots < y_m$ there is an automorphism $f : \mathbb{Q} \rightarrow \mathbb{Q}$ with $f(x_i) = y_i$ for $i \leq m$. Again there might be a number of automatic variations. Call an automatic copy of \mathbb{Q} *automatically homogeneous* if for every two tuples there is an automorphism as above that is also regular.

Proposition 1.5.7 [93] *There is an automatic copy of \mathbb{Q} that is automatically homogeneous. There is an automatic copy of \mathbb{Q} that is not automatically homogeneous.*

Scott ranks

Every countable structure \mathcal{A} has a sentence of the infinitary logic $L_{\omega_1, \omega}$ (it allows, in addition to FO, countable disjuncts but still only finitely many free variables) that characterises \mathcal{A} up to isomorphism. The *Scott rank* of \mathcal{A} is the minimal quantifier rank amongst all such sentences.

Theorem 1.5.8 ([86]) *For every computable ordinal there is an automatic structure of Scott Rank at least α .*

The idea is to massage the configuration space of Turing machines presenting a computable structure (having Scott Rank α) to get an automatic structure of similar rank.

1.5.2 On the universal word-automatic structure

We conclude by highlighting some model-theoretic properties of the universal structure $\mathcal{S}_{[2]}$.

- (i) $\mathcal{S}_{[2]}$ has infinite VC-dimension [15]. That is, there is a formula $\phi(x, z)$ that defines a family of sets of the form $\phi(-, z)^{\mathcal{S}_{[2]}}$ as one varies the parameter z , and this family fully shatters arbitrarily large finite sets.
- (ii) $\mathcal{S}_{[2]}$ admits quantifier elimination (QE) in the expansion of all definable unary predicates and binary functions. In fact, no expansion with definable unary functions (and arbitrary predicates) admits QE [15].

Blumensath [20, p. 67] raised the question of whether there are non-standard models of the theory of the universal structure $\mathcal{S}_{[2]}$ in S-AutStr . Here we sketch an argument resting on Theorem 1.4.26 that shows that there are no word-automatic non-standard models. This result was obtained in discussions with Bakhadyr Khoussainov.

Theorem 1.5.9 $\mathcal{S}_{[2]}$ is the only word-automatic model of its theory.

Proof Assume, for a contradiction, an automatic presentation of a non-standard elementary extension of $\mathcal{S}_{[2]}$. By ‘component’ we mean a maximal set of elements connected by successor relations. Every elementary extension of $\mathcal{S}_{[2]}$ consists of the standard component isomorphic to $\mathcal{S}_{[2]}$ (containing the root), and any number of non-standard components, that are, as unlabelled graphs, all isomorphic to one-another. The non-standard components are distinguished by the infinite sequences of 0-1 successors ascending towards the root.

(0) *The set of representatives of elements of each component is regular.*

Indeed, the equivalence relation of belonging to the same component is $\text{FO} + \exists^\infty$ -definable in the model (by saying that there is a common ancestor having finite distance from both elements), hence regular in the representation.

(1) *There is a non-standard element below every standard node.*

This follows from the fact that the formula

$$\forall x, x', y : \text{el}(x, x') \wedge x \prec y \rightarrow \exists y' : \text{el}(y, y') \wedge x' \prec y'$$

being true in $\mathcal{S}_{[2]}$ must also hold in every non-standard model.

Combining observation (0) and Theorem 1.4.26 we may assume that the presentation restricted to the standard component is the natural one having the identity as naming function. The binary ω -sequence naturally associated with an infinite branch of the standard component provides a representation of the set of nodes along that branch consistent with the assumed presentation of the model. Denote by Π the set of paths with a non-standard element below them.

(2) *The set Π is ω -regular.*

Indeed, a Büchi-automaton is built to guess a finite word representing a non-standard element and to check, using the automata of the assumed presentation, that it is a descendant of all finite prefixes of the input path. Given that our model is countable, hence so is Π , we have the following consequence of claim (2).

(3) *Every path in Π is ultimately periodic with a period of bounded length.*

To close the circle, consider for each $n \in \mathbb{N}$ the sentence

$$\forall x \exists y |y| > |x| \wedge 0^n 1 \preceq_{\text{prefix}} y \wedge (\forall z \prec_{\text{prefix}} y)[\text{end}_1(z) \rightarrow z 0^n 1 \preceq_{\text{prefix}} y]$$

where $\text{end}_1(z)$ is shorthand for saying that the last letter of z is 1. This sentence expresses that for every length $|x|$ there is a longer word y with as many initial prefixes in $(0^n 1)^*$ as possible. In particular this sentence holds for non-standard elements x . Consequently,

(4) *for every $n \in \mathbb{N}$ there is an infinite branch of the standard component with label $(0^n 1)^\omega$ and having non-standard elements below it.*

This contradicts observation (3). \triangleleft

Therefore, by Theorem 1.4.20, there are no countable ω -word automatic non-standard models either. Furthermore, using Theorem 1.4.29 in place of Theorem 1.4.26 in the argument shows there are no non-standard finite-tree automatic models of $S_{[2]}$. To prove that there are no uncountable ω -word automatic non-standard models of $S_{[2]}$ one tightens (4) and exploits that all automatic presentations of non-standard components are equivalent.

References

- [1] K. Aehlig, J. G. de Miranda, and C.-H. L. Ong. Safety is not a restriction at level 2 for string languages. In *FoSSaCS*, pages 490–504, 2005.
- [2] J.-P. Allouche and J. Shallit. *Automatic Sequences, Theory, Applications, Generalizations*. Cambridge University Press, 2003.
- [3] J.-P. Allouche and J. O. Shallit. The Ubiquitous Prouhet-Thue-Morse Sequence. In C. Ding, T. Helleseth, and H. Niederreiter, editors, , pages 1–16. Springer-Verlag, 1999.
- [4] V. Bárány. A hierarchy of automatic ω -words having a decidable MSO theory. *Journées Montoisées '06*, Rennes, 2006.
- [5] V. Bárány. Invariants of automatic presentations and semi-synchronous transductions. In *STACS '06*, volume 3884 of *LNCS*, pages 289–300, 2006.
- [6] V. Bárány. *Automatic Presentations of Infinite Structures*. Phd thesis, RWTH Aachen University, 2007.
- [7] V. Bárány. A hierarchy of automatic ω -words having a decidable MSO theory. *R.A.I.R.O. Theoretical Informatics and Applications*, 42:417–450, 2008.
- [8] V. Bárány. Semi-synchronous transductions. *Acta Informatica*, 46(1):29–42, 2009.
- [9] V. Bárány, L. Kaiser, and A. Rabinovich. Eliminating cardinality quantifiers from MLO. Manuscript, 2007.
- [10] V. Bárány, Ch. Löding, and O. Serre. Regularity problems for visibly pushdown languages. In *STACS '06*, volume 3884 of *LNCS*, pages 420–431, 2006.
- [11] K. Barthelmann. On equational simple graphs. Tech. Rep. 9, Universität Mainz, Institute für Informatik, 1997.
- [12] K. Barthelmann. When can an equational simple graph be generated by hyperedge replacement? In *MFCS*, pages 543–552, 1998.
- [13] M.-P. Béal and D. Perrin. Symbolic Dynamics and Finite Automata. In A. Salomaa and G. Rosenberg, editors, *Handbook of Formal Languages, Vol. 2*, pages 463–503. Springer Verlag, 1997.

- [14] M. Benedikt and L. Libkin. Tree extension algebras: logics, automata, and query languages. In *Proceedings of the 17th Annual IEEE Symposium on Logic in Computer Science (LICS)*, pages 203–212, 2002.
- [15] M. Benedikt, L. Libkin, Th. Schwentick, and L. Segoufin. A model-theoretic approach to regular string relations. In Joseph Halpern, editor, *LICS 2001*, pages 431–440. IEEE Computer Society, June 2001.
- [16] M. Benedikt, L. Libkin, Th. Schwentick, and L. Segoufin. Definable relations and first-order query languages over strings. *J. ACM*, 50(5):694–751, 2003.
- [17] D. Berwanger and A. Blumensath. The monadic theory of tree-like structures. In E. Grädel, W. Thomas, and T. Wilke, editors, *Automata, Logics, and Infinite Games*, number 2500 in LNCS, chapter 16, pages 285–301. Springer Verlag, 2002.
- [18] A. Bès. Undecidable extensions of Büchi arithmetic and Cobham-Seménov theorem. *Journal of Symbolic Logic*, 62(4):1280–1296, 1997.
- [19] A. Bès. An Extension of the Cobham-Seménov Theorem. *J. of Symb. Logic*, 65(1):201–211, 2000.
- [20] A. Blumensath. Automatic Structures. Diploma thesis, RWTH-Aachen, 1999.
- [21] A. Blumensath. Prefix-Recognisable Graphs and Monadic Second-Order Logic. Technical report AIB-2001-06, RWTH Aachen, 2001.
- [22] A. Blumensath. Axiomatising Tree-interpretable Structures. In *STACS*, volume 2285 of *LNCS*, pages 596–607. Springer-Verlag, 2002.
- [23] A. Blumensath, Th. Colcombet, and Ch. Löding. Logical theories and compatible operations. In J. Flum, E. Grädel, and T. Wilke, editors, *Logic and Automata: History and Perspectives*, Texts in Logic and Games, pages 73–106. Amsterdam University Press, 2007.
- [24] A. Blumensath and E. Grädel. Automatic structures. In *LICS 2000*, pages 51–62. IEEE Computer Society, 2000.
- [25] A. Blumensath and E. Grädel. Finite presentations of infinite structures: Automata and interpretations. *Theory of Comp. Sys.*, 37:641 – 674, 2004.
- [26] V. Bruyère, G. Hansel, Ch. Michaux, and R. Villemaire. Logic and p-recognizable sets of integers. *Bull. Belg. Math. Soc.*, 1:191 – 238, 1994.
- [27] J. R. Büchi. Weak second-order arithmetic and finite automata. *Zeit. Math. Logik Grund. Math.*, 6:66–92, 1960.
- [28] A. J. Cain, E. F. Robertson, and N. Ruskuc. Subsemigroups of groups: presentations, malcev presentations, and automatic structures. *Journal of Group Theory*, 9(3):397–426, 2006.
- [29] C. M. Campbell, E. F. Robertson, N. Ruskuc, and R. M. Thomas. Automatic semigroups. *Theor. Comput. Sci.*, 250(1-2):365–391, (2001).
- [30] C. M. Campbell, E. F. Robertson, N. Ruskuc, and R. M. Thomas. Automatic completely-simple semigroups. *Acta Math. Hungar.*, 96:201–215, 2002.
- [31] J.W. Cannon, D.B.A. Epstein, D.F. Holt, S.V.F. Levy, M.S. Paterson, and W.P. Thurston. *Word processing in groups*. Jones and Barlett Publ., Boston, MA, 1992.

- [32] A. Carayol. Regular sets of higher-order pushdown stacks. In *Proceedings of Mathematical Foundations of Computer Science (MFCS 2005)*, volume 3618 of *LNCS*, pages 168–179, 2005.
- [33] A. Carayol and Th. Colcombet. On equivalent representations of infinite structures. In *ICALP*, volume 2719 of *LNCS*, pages 599–610. Springer, 2003.
- [34] A. Carayol and A. Meyer. Linearly bounded infinite graphs. In *MFCS*, volume 3618 of *Lecture Notes in Computer Science*, pages 180–191. Springer, 2005.
- [35] A. Carayol and A. Meyer. Context-Sensitive Languages, Rational Graphs and Determinism. *Logical Methods in Computer Science*, 2(2), 2006.
- [36] A. Carayol and C. Morvan. On rational trees. In Z. Ésik, editor, *CSL 06*, volume 4207 of *LNCS*, pages 225–239, 2006.
- [37] A. Carayol and S. Wöhrle. The Caucal hierarchy of infinite graphs in terms of logic and higher-order pushdown automata. In *FSTTCS*, volume 2914 of *LNCS*, pages 112–123. Springer, 2003.
- [38] O. Carton and W. Thomas. The monadic theory of morphic infinite words and generalizations. *Information and Computation*, 176(1):51–65, 2002.
- [39] A. Caryol and Ch. Löding. MSO on the Infinite Binary Tree: Choice and Order. In *CSL*, volume 4646 of *LNCS*, pages 161–176, 2007.
- [40] D. Caucal. Monadic theory of term rewritings. In *LICS*, pages 266–273. IEEE Computer Society, 1992.
- [41] D. Caucal. On the regular structure of prefix rewriting. *Theor. Comput. Sci.*, 106(1):61–86, 1992.
- [42] D. Caucal. On infinite transition graphs having a decidable monadic theory. In *ICALP'96*, volume 1099 of *LNCS*, pages 194–205, 1996.
- [43] D. Caucal. On infinite terms having a decidable monadic theory. In *MFCS*, pages 165–176, 2002.
- [44] D. Caucal. Deterministic graph grammars. In J. Flum, E. Grädel, and T. Wilke, editors, *Logic and Automata: History and Perspectives*, Texts in Logic and Games, pages 169–250. Amsterdam University Press, 2007.
- [45] D. Cenzer and J.B. Remmel. Complexity-theoretic model theory and algebra. In *Handbook of Recursive Mathematics, Vol. 1*, volume 138 of *Studies in Logic and the Foundations of Mathematics*, pages 381–513. North-Holland, Amsterdam, 1998.
- [46] Ch. Choffrut. A short introduction to automatic group theory, 2002.
- [47] T. Colcombet and C. Löding. Transforming structures by set interpretations. *Logical Methods in Computer Science*, 3(2), 2007.
- [48] Th. Colcombet. On families of graphs having a decidable first order theory with reachability. In *ICALP*, volume 2380 of *LNCS*, pages 98–109. Springer, 2002.
- [49] Th. Colcombet. Equational presentations of tree-automatic structures. In Workshop on Automata, Structures and Logic, Auckland, NZ, 2004.
- [50] Th. Colcombet. *Propriétés et représentation de structures infinies*. Thèse de doctorat, Université Rennes I, 2004.

- [51] Th. Colcombet. A combinatorial theorem for trees. In *ICALP*, volume 4596 of *LNCS*, pages 901–912. Springer, 2007.
- [52] H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree Automata Techniques and Applications. In preparation, draft available online at <http://www.grappa.univ-lille3.fr/tata/>.
- [53] B. Courcelle. *Graph algebras and monadic second-order logic*. Cambridge University Press, in writing...
- [54] B. Courcelle. The definability of equational graphs in monadic second-order logic. In *ICALP*, volume 372 of *LNCS*, pages 207–221. Springer, 1989.
- [55] B. Courcelle. Graph rewriting: An algebraic and logic approach. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics*, pages 193–242. Elsevier and MIT Press, 1990.
- [56] B. Courcelle. Recursive applicative program schemes. In J. v.d. Leeuwen, editor, *Handbook of Theoretical Computer Science, Vol. B*, pages 459–492. Elsevier and MIT Press, 1990.
- [57] B. Courcelle. The monadic second-order logic of graphs ix: Machines and their behaviours. *Theoretical Computer Science*, 151(1):125–162, 1995.
- [58] B. Courcelle. Finite model theory, universal algebra and graph grammars. In *LFCS '97, Proceedings of the 4th International Symposium on Logical Foundations of Computer Science*, pages 53–55, London, UK, 1997. Springer-Verlag.
- [59] B. Courcelle. The Expression of Graph Properties and Graph Transformations in Monadic Second-Order Logic. In G. Rozenberg, editor, *Handbook of graph grammars and computing by graph transformations, vol. 1: Foundations*, pages 313–400. World Scientific, New-Jersey, London, 1997.
- [60] B. Courcelle and J. A. Makowsky. Fusion in Relational Structures and the Verification of Monadic Second-Order Properties. *Mathematical Structures in Computer Science*, 12(2):203–235, 2002.
- [61] B. Courcelle and I. Walukiewicz. Monadic second-order logic, graph coverings and unfoldings of transition systems. *Annals of Pure and Applied Logic*, 92:35–62, 1998.
- [62] W. Damm. The IO- and OI hierarchies. *Theoretical Computer Science*, 20(2):95–208, 1982.
- [63] C. Delhommé. Automaticité des ordinaux et des graphes homogènes. *Comptes Rendus Mathematique*, 339(1):5–10, 2004.
- [64] M.J. Dunwoody. The accessibility of finitely presented groups. *Inventiones Mathematicae*, 81(3):449–457, 1985.
- [65] S. Eilenberg, C.C. Elgot, and J.C. Shepherdson. Sets recognised by n -tape automata. *Journal of Algebra*, 13(4):447–464, 1969.
- [66] C. C. Elgot and J. E. Mezei. On relations defined by generalized finite automata. *IBM J. Research and Development*, 9:47 – 68, 1965.

- [67] C. C. Elgot and M. O. Rabin. Decidability and undecidability of extensions of second (first) order theory of (generalized) successor. *Journal of Symbolic Logic*, 31(2):169–181, 1966.
- [68] C.C. Elgot. Decision problems of finite automata design and related arithmetics. *Trans. Amer. Math. Soc.*, 98:21–51, 1961.
- [69] J. Engelfriet. Context-free graph grammars. In *Handbook of formal languages, vol. III*, pages 125–213. Springer-Verlag New York, Inc., New York, NY, USA, 1997.
- [70] Y.L. Ershov, S.S. Goncharov, A. Nerode, and J.B. Remmel, editors. *Handbook of Recursive Mathematics, Vol. 1*, volume 138 of *Studies in Logic and the Foundations of Mathematics*. North-Holland, Amsterdam, 1998.
- [71] B. Farb. Automatic Groups: A Guided Tour. *L'Enseignement Math.*, 38:291–313, 1992.
- [72] S. Fratani. *Automates à Piles de Piles ... de Piles*. Thèse de doctorat, Université Bordeaux 1, 2005.
- [73] S. Fratani. Regular sets over tree structures. Rapport Interne 1358-05, LaBRI, Université Paris 7, 2005.
- [74] S. Fratani. The theory of successor extended by severals predicates. *Journées Montoises '06*, Rennes, 2006.
- [75] S. Fratani and G. Sézergues. Iterated pushdown automata and sequences of rational numbers. *Ann. Pure Appl. Logic*, 141(3):363–411, 2006.
- [76] Ch. Frougny. Numeration systems. In M. Lothaire, editor, *Algebraic Combinatorics on Words*. Cambridge University Press, 2002.
- [77] E. Grädel. Simple interpretations among complicated theories. *Information Processing Letters*, 35:235–238, 1990.
- [78] E. Grädel, P. G. Kolaitis, L. Libkin, M. Marx, J. Spencer, M. Vardi, Y. Venema, and S. Weinstein. *Finite Model Theory and Its Applications*. Springer-Verlag, 2007.
- [79] E. Grädel, W. Thomas, and T. Wilke, editors. *Automata, Logics, and Infinite Games*, volume 2500 of *LNCS*. Springer-Verlag, 2002.
- [80] A. Habel. *Hyperedge Replacement: Grammars and Languages*, volume 643 of *Lecture Notes in Computer Science*. Springer, 1992.
- [81] M. Hague, A.S. Murawski, C.-H. L. Ong, and O. Serre. Collapsible pushdown automata and recursion schemes. In *LICS'08*. IEEE Computer Society, 2008.
- [82] G. Hjorth, B. Khoussainov, A. Montalbán, and A. Nies. From automatic structures to Borel structures. In *23rd Symposium on Logic in Computer Science (LICS)*, 2008.
- [83] M. Hoffmann, D. Kuske, F. Otto, and R. M. Thomas. Some relatives of automatic and hyperbolic groups, 2002.
- [84] M. Hoffmann and R. M. Thomas. Notions of automaticity in semigroups. *Semigroup Forum*, 66:337–367., (2003).
- [85] L. Kaiser, S. Rubin, and V. Bárány. Cardinality and counting quantifiers on ω -automatic structures. In *STACS '08*, volume 08001 of *Dagstuhl Seminar Proceedings*, pages 385–396. Internationales Begegnungs- und

- Forschungszentrum fuer Informatik (IBFI), Schloss Dagstuhl, Germany, 2008.
- [86] B. Khoussainov and M. Minnes. Model theoretic complexity of automatic structures. *Annals of Pure and Applied Logic*, To appear, 2008.
 - [87] B. Khoussainov and A. Nerode. Automatic presentations of structures. In *LCC '94*, volume 960 of *LNCS*, pages 367–392. Springer-Verlag, 1995.
 - [88] B. Khoussainov, A. Nies, S. Rubin, and F. Stephan. Automatic structures: Richness and limitations. In *LICS'04*, pages 44–53, 2004.
 - [89] B. Khoussainov and S. Rubin. Graphs with automatic presentations over a unary alphabet. *Journal of Automata, Languages and Combinatorics*, 6(4):467–480, 2001.
 - [90] B. Khoussainov, S. Rubin, and F. Stephan. Definability and regularity in automatic structures. In *STACS '04*, volume 2996 of *LNCS*, pages 440–451, 2004.
 - [91] B. Khoussainov, S. Rubin, and F. Stephan. Automatic linear orders and trees. *ACM Transactions on Computational Logic*, 6(4):675–700, 2005.
 - [92] T. Knapik, D. Niwinski, and P. Urzyczyn. Higher-order pushdown trees are easy. In *FoSSaCS'02*, volume 2303 of *LNCS*, pages 205–222, 2002.
 - [93] D. Kuske. Is cantor's theorem automatic? In *LPAR*, volume 2850 of *LNCS*, pages 332–345. Springer, 2003.
 - [94] D. Kuske and M. Lohrey. First-order and counting theories of ω -automatic structures. In *FoSSaCS*, pages 322–336, 2006.
 - [95] D. Kuske and M. Lohrey. Automatic structures of bounded degree revisited. arXiv:0810.4998, 2008.
 - [96] D. Kuske and M. Lohrey. Hamiltonicity of automatic graphs. In *FIP TCS 2008*, 2008.
 - [97] H. Lauchli and Ch. Savioz. Monadic Second Order Definable Relations on the Binary Tree. *J. of Symbolic Logic*, 52(1):219–226, 1987.
 - [98] S. Lifsches and S. Shelah. Uniformization and skolem functions in the class of trees. *Journal of Symbolic Logic*, 63:103–127, 1998.
 - [99] Ch. Löding. *Infinite Graphs Generated by Tree Rewriting*. Doctoral thesis, RWTH Aachen, 2003.
 - [100] Christof Löding. Reachability problems on regular ground tree rewriting graphs. *Theor. Comp. Sys.*, 39(2):347–383, 2006.
 - [101] M. Lohrey. Automatic structures of bounded degree. In *LPAR*, volume 2850 of *LNCS*, pages 346–360. Springer, 2003.
 - [102] M. Lohrey. Decidability and complexity in automatic monoids. In *Developments in Language Theory*, pages 308–320, 2004.
 - [103] A. Meyer. Traces of term-automatic graphs. *R.A.I.R.O. Theoretical Informatics and Applications*, 42, 2008.
 - [104] C. Michaux and F. Point. Les ensembles k -reconnaissables sont définissables dans $\langle \mathbb{N}, +, V_k \rangle$. *C. R. Acad. Sci. Paris Sér. I Math.*, 303(19):939–942, 1986.
 - [105] Ch. Morvan. *Les graphes rationnels*. Thèse de doctorat, Université de Rennes 1, Novembre 2001.
 - [106] Ch. Morvan. Classes of rational graphs. *Journées Montoises '06*, Rennes, 2006.

- [107] Ch. Morvan and Ch. Rispal. Families of automata characterizing context-sensitive languages. *Acta Informatica*, 41(4-5):293–314, 2005.
- [108] Ch. Morvan and C. Stirling. Rational graphs trace context-sensitive languages. In A. Pultr and J. Sgall, editors, *MFCS'01*, volume 2136 of *LNCS*, pages 548–559, 2001.
- [109] A. A. Muchnik. The definable criterion for definability in Presburger arithmetic and its applications. *Theor. Comput. Sci.*, 290(3):1433–1444, 2003.
- [110] D. E. Muller and P. E. Schupp. Context-free languages, groups, the theory of ends, second-order logic, tiling problems, cellular automata, and vector addition systems. *Bull. Amer. Math. Soc.*, 4(3):331–334, 1981.
- [111] D. E. Muller and P. E. Schupp. Groups, the theory of ends, and context-free languages. *J. Comput. Syst. Sci.*, 26(3):295–310, 1983.
- [112] D. E. Muller and P. E. Schupp. The theory of ends, pushdown automata, and second-order logic. *Theor. Comput. Sci.*, 37:51–75, 1985.
- [113] A. A. Nabebin. Expressibility in a restricted second-order arithmetic. *Siberian Mathematical Journal*, 18(4):588–593, 1977.
- [114] A. Nies. Describing groups. *Bulletin of Symbolic Logic*, 13(3):305–339, 2007.
- [115] D. Niwiński. On the cardinality of sets of infinite trees recognizable by finite automata. In *Proceedings of the 16th International Symposium on Mathematical Foundations of Computer Science, MFCS'91*, volume 520, pages 367–376. Springer, 1991.
- [116] G. P. Oliver and R. M. Thomas. Finitely generated groups with automatic presentations. In *STACS 2005*, volume 3404 of *LNCS*, pages 693–704. Springer, 2005.
- [117] C.-H. L. Ong. On model-checking trees generated by higher-order recursion schemes. In *LICS*, pages 81–90. IEEE Computer Society, 2006.
- [118] J.-J. Pansiot. On various classes of infinite words obtained by iterated mappings. In *Automata on Infinite Words*, pages 188–197, 1984.
- [119] A. Rabinovich. On decidability of monadic logic of order over the naturals extended by monadic predicates. Unpublished note, 2005.
- [120] A. Rabinovich and W. Thomas. Decidable theories of the ordering of natural numbers with unary predicates. Submitted, 2006.
- [121] M. Rigo. Numeration systems on a regular language: Arithmetic operations, recognizability and formal power series. *Theoretical Computer Science*, 269:469, 2001.
- [122] M. Rigo and A. Maes. More on generalized automatic sequences. *J. of Automata, Languages and Combinatorics*, 7(3):351–376, 2002.
- [123] Ch. Rispal. The synchronized graphs trace the context-sensistive languages. *Electronic Notes in Theor. Comp. Sci.*, 68(6), 2002.
- [124] S. Rubin. *Automatic Structures*. Phd thesis, University of Auckland, NZ, 2004.
- [125] S. Rubin. Automata presenting structures: A survey of the finite-string case. *Bulletin of Symbolic Logic*, 14(2):169–209, 2008.

- [126] A. L. Semenov. Decidability of monadic theories. In *Mathematical Foundations of Computer Science, Prague, 1984*, volume 176 of *LNCS*, page 162?175. Springer, Berlin, 1984.
- [127] G. Sénizergues. Semi-groups acting on context-free graphs. In *ICALP '96: Proceedings of the 23rd International Colloquium on Automata, Languages and Programming*, pages 206–218, London, UK, 1996. Springer-Verlag.
- [128] G. Sénizergues. The bisimulation problem for equational graphs of finite out-degree. *SIAM J. Comput.*, 34(5):1025–1106, 2005.
- [129] P. V. Silva and B. Steinberg. A geometric characterization of automatic monoids. *The Quarterly Journal of Mathematics*, 55:333–356, 2004.
- [130] J. Su and S. Grumbach. Finitely representable databases (extended abstract). In *In Proc. 13th ACM Symp. on Principles of Database Systems*, 1994.
- [131] A. Szilard, Sh. Yu, K. Zhang, and J. Shallit. Characterizing regular languages with polynomial densities. In *MFCS*, pages 494–503, 1992.
- [132] W. Thomas. Automata on infinite objects. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics*, pages 133–192. Elsevier and MIT Press, 1990.
- [133] W. Thomas. Languages, automata, and logic. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages, volume III*, pages 389–455. Springer, New York, 1997.
- [134] W. Thomas. Constructing Infinite Graphs with a Decidable MSO-Theory. In *MFCS*, volume 2747 of *LNCS*, pages 113–124, 2003.
- [135] B.A. Trahtenbrot. Finite automata and the logic of one-place predicates. Russian. *Siberian Mathematical Journal*, 3:103–131, 1962. English translation: American Mathematical Society Translations, Series 2, 59 (1966), 23–55.
- [136] T. Tsankov. The additive group of the rationals is not automatic. manuscript, 2009.
- [137] R. Villemaire. The theory of $\langle \mathbb{N}, +, V_k, V_l \rangle$ is undecidable. *Theoretical Computer Science*, 106:337–349, 1992.
- [138] I. Walukiewicz. Monadic second-order logic on tree-like structures. *Theoretical Computer Science*, 275:311–346, 2002.
- [139] S. Wöhrle and W. Thomas. Model checking synchronized products of infinite transition systems. In *LICS '04*, pages 2–11, Washington, DC, USA, 2004. IEEE Computer Society.

A Myhill-Nerode theorem for automata with advice

Alex Kruckman

University of California, Berkeley

kruckman@gmail.com

Sasha Rubin

TU Vienna and IST Austria

sasha.rubin@gmail.com

John Sheridan

jhs223@cornell.edu

Ben Zax

benzax@gmail.com

An automaton with advice is a finite state automaton which has access to an additional fixed infinite string called *an advice tape*. We refine the Myhill-Nerode theorem to characterize the languages of finite strings that are accepted by automata with advice. We do the same for tree automata with advice.

1 Introduction

Consider an extension of the classical model of finite automata operating on finite strings in which the machine simultaneously reads a fixed *advice tape* — an infinite string A . A *deterministic finite-string automaton with advice A* is like a deterministic finite-string automaton, except that at step n the next state depends on the current state, the n^{th} symbol of the input $w \in \Sigma^*$, and the n^{th} symbol of $A \in \Gamma^\omega$. The automaton halts once all of w has been read and accepts w if and only if it is in a final state (in the *non-terminating* model of Section 4, the automaton reads the rest of the advice tape and accepts according to a Muller condition).

We now give a formal definition.

Definition 1. An automaton with advice is a tuple $M = (Q, \Sigma, \Gamma, A, \delta, q_0, F)$.

1. Q is the finite set of states of the automaton.
2. Σ is a finite set of symbols called the input alphabet.
3. Γ is a finite set of symbols called the advice alphabet.
4. $A \in \Gamma^\omega$ is the advice string.
5. $\delta : Q \times \Gamma \times \Sigma \rightarrow Q$ is the transition function.
6. $q_0 \in Q$ is the initial state.
7. $F \subseteq Q$ is the acceptance condition.

The run of M on a string $w \in \Sigma^*$ is a sequence of states $\alpha \in Q^{|w|+1}$ such that $\alpha_0 = q_0$ and for $1 \leq n \leq |w|$, $\alpha_n = \delta(\alpha_{n-1}, A_n, w_n)$.

We say a string $w \in \Sigma^*$ is accepted by the automaton with advice M if the final state appearing the run of M on w is in F .

A language $L \subseteq \Sigma^*$ is regular with advice A if it is the language accepted by some automaton with advice A . A language L is regular with advice if there exists A such that L is regular with advice A . Thus L is not regular with advice means that there is no A such that L is regular with advice A .

What is the power of this model of computation? We make some trivial observations:

1. Every regular language is regular with advice (indeed, any advice will do).
2. Every unary language L (ie. subset of $\{1\}^*$) is regular with advice (indeed, let A be the characteristic sequence of L).

Is every language regular with advice? For a fixed advice A the answer is clearly ‘no’: there are continuum many languages and only countably many languages regular with advice A (since there are countably many finite automata).

This simple argument does not preclude the possibility that for every language L there is an advice A_L such that L is regular with advice A_L . Also note that the standard pumping and Myhill-Nerode arguments showing non-regularity do not apply in the presence of advice. We now sketch an alternative argument that in fact the language $\{0^n 1^n : n \in \mathbb{N}\}$ is not regular with advice.

Recall that for a language L , the equivalence relation \equiv_L on Σ^* , called the Myhill-Nerode congruence, is defined as follows: $x \equiv_L y$ if for all $z \in \Sigma^*$ it holds that $xz \in L \iff yz \in L$. The classical Myhill-Nerode theorem states:

Theorem 2 (Myhill-Nerode). *A language $L \subseteq \Sigma^*$ is regular if and only if \equiv_L has finitely many equivalence classes.*

The proof of the Myhill-Nerode theorem for classical automata suggests the following observation regarding automata with advice:

Let M be an automaton with advice which accepts the language L with some advice A . Suppose that M , starting in the initial state, reaches the same state on input x as on input y . If M is also at the same place in the advice tape A after reading x and y , that is, if x and y have the same length, then $x \equiv_L y$. Thus, for every n , the number of classes of \equiv_L restricted to Σ^n (the strings in Σ^* of length exactly n) is at most the number of states in M .

For language L and integer n write $\equiv_{L,n}$ for the equivalence relation \equiv_L restricted to Σ^n . If there exists n such that the number of equivalence classes of $\equiv_{L,n}$ is k , then no automaton with advice having fewer than k states accepts L .

We now have a way to prove that certain languages are not regular with advice. Consider $L := \{0^n 1^n : n \in \mathbb{N}\}$ and note that for every n , no pair of strings in the set $X_n := \{0^a 1^{n-a} : \frac{n}{2} \leq a \leq n\}$ are $\equiv_{L,n}$ -equivalent. But the size of X_n is unbounded as n grows. Thus L is not regular with advice.

The observation above gives one direction of a Myhill-Nerode like characterization. We prove the other direction in Section 2. The role of the size of the alphabet Γ is considered in Section 3. A variation of the model — which we call *non-terminating* — in which the automaton reads the rest of the advice is considered in Section 4. Finally, we mention in Section 5 that the results go through for tree automata with a fixed infinite tree as advice.

Related work

Automata over finite words can be identified with weak monadic second-order formulas over the structure $(\mathbb{N}, \text{succ})$. Non-terminating automata with advice correspond to WMSO formulas over expansions of $(\mathbb{N}, \text{succ})$ by unary predicates \bar{P} (this is implicit in [4, 2, 8, 1]). Questions of logical decidability are equivalent to the \bar{P} -acceptance problem: given a Muller automaton M , decide whether or not M accepts \bar{P} . Similar things are done for automata operating on finite trees [5][Definition 11] and [3].

It is easy to see that the languages recognized by non-terminating automata with advice are closed under logical operations such as union, complementation, projection, permutation of co-ordinates and instantiation (see for instance [5]). Consequently, one may define *automatic structures with advice* ([3]). These are relational structures whose domain and atomic relations are recognized by automata with

advice. The case without advice is well studied, and such structures have decidable first-order theory (see [9]). The main programme there has been to supply techniques for showing non-automaticity. For instance, there is a difficult proof of the fact that $(\mathbb{Q}, +)$ is not automatic without advice ([10]). However it is automatic with advice (communicated by Frank Stephan and Joe Miller, and reported in [7]). Because this example is not yet well known we present it here.

Example 3. *To simplify exposition, we give a presentation of $([0, 1) \cap \mathbb{Q}, +)$. Each rational is coded by a finite string over the alphabet $\{0, 1, \#\}$. Automata for the domain and the addition will have access to the advice string*

$$A = 10\#11\#100\#101\#110\#111\#1000\#\dots$$

which is a version of the Champernowne-Smarandache string. To every rational q in $[0, 1)$ there is a unique finite sequence of integers $a_1 \cdots a_n$ such that $0 \leq a_i < i$, $q = \sum_{i=2}^n \frac{a_i}{i!}$, and n is minimal. The presentation codes this rational as $f(a_2)\#f(a_3)\#f(a_4)\cdots\#f(a_n)$ where f sends a_i to the binary string of length $\lceil \log_2 i \rceil + 1$ representing a_i . Addition $a + b$ is performed least significant digit first (right to left) based on the fact that

$$\frac{a_i + b_i + c}{i!} = \frac{1}{(i-1)!} + \frac{a_i + b_i + c - i}{i!}$$

where $c \in \{0, 1\}$ is the carry in. In other words, if $a_i + b_i + c \geq i$ then write $a_i + b_i + c - i$ in the i th segment and carry a 1 into the $(i-1)$ st segment; and if $a_i + b_i + c < i$ then write this under the i th segment and carry a 0 into the $(i-1)$ st segment. These comparisons and additions can be performed since the advice tape is storing i in the same segment as a_i and b_i .

We remark that the advice string A above has decidable acceptance problem ([1]). Consequently every structure that is automatic with this advice has decidable first-order theory.

We end with a question that we hope will spur interest: what are other interesting examples of structures that are automatic with advice?

Acknowledgements

Research supported by the National Science Foundation through the Research Experiences for Undergraduates (REU) program at Cornell University, the European Science Foundation for the activity entitled ‘Games for Design and Verification’, FWF NFN Nr. S11407-N23 (RiSE), ERC 279307: Graph Games, PROSEED WWTF Nr. ICT 10-050, and ARISE FWF Nr. S11403-N23.

2 The Myhill-Nerode Theorem

Let Σ be a finite alphabet and $L \subseteq \Sigma^*$ a language. Define an equivalence relation $\equiv_{L,n}$ on Σ^n , the set of all strings of length n , by $x \equiv_{L,n} y$ if for all $z \in \Sigma^*$ it holds that $xz \in L \iff yz \in L$. This is the usual Myhill-Nerode congruence restricted to strings of length n .

Theorem 4 (Myhill-Nerode theorem with advice). *A language $L \subseteq \Sigma^*$ is regular with advice if and only if there is some $k \in \mathbb{N}$ such that for every $n \geq 0$, $\equiv_{L,n}$ has at most k equivalence classes.*

Proof. Suppose L is regular with advice. Let $M = (Q, \Sigma, \Gamma, A, \delta, q_0, F)$ be the automaton recognizing L , and let $k = |Q|$. Now assume that for some n , $\equiv_{L,n}$ divides the strings of length n into l equivalence classes, with $l > k$. Pick representative strings x_1, \dots, x_l in these classes. Let $q_i \in Q$ be the $(n+1)^{\text{st}}$ state in the run of M on x_i , that is, the state reached after reading the final character of x_i . Since $l > |Q|$, $q_i = q_j$

for some $i \neq j$. Then for all $z \in \Sigma^*$, M accepts $x_i z$ if and only if M accepts $x_j z$, since the run of M on these two strings is the same after stage n . This contradicts the assumption that x_i and x_j are representatives of distinct $\equiv_{L,n}$ -classes.

Conversely, suppose we have such a bound k . For each $n \in \mathbb{N}$, let \mathcal{C}_n be the collection of equivalence classes of $\equiv_{L,n}$, so $|\mathcal{C}_n| \leq k$.

We must construct an automaton M recognizing L . Let $Q = \{1, \dots, 2k\}$ be the set of states, and let $F = \{1, \dots, k\} \subset Q$. The idea is that we have enough room to represent each equivalence class by an accepting or a rejecting state as necessary at each stage. Set $q_0 = 1$ if the empty string is in L and $q_0 = k + 1$ otherwise.

For $x, y \in \Sigma^n$, if $x \equiv_{L,n} y$, then $x \in L$ if and only if $y \in L$ (appending the empty string as a suffix). Thus, \mathcal{C}_n is partitioned into those classes which are “accepting” and those which are not. Also if $x \equiv_{L,n} y$, then for all $a \in \Sigma$, $xa \equiv_{L,n+1} ya$, since for all $z \in \Sigma^*$, $xaz \in L$ if and only if $yaz \in L$. This defines a function $h_n : \mathcal{C}_n \times \Sigma \rightarrow \mathcal{C}_{n+1}$ so that if $C \in \mathcal{C}_n$ and x is a string in C , then xa is a string in the class $h_n(C, a)$.

For each $n > 0$, identify the “accepting” classes with states from $\{1, \dots, k\}$ and the remaining classes with states from $\{k + 1, \dots, 2k\}$. The remaining work is to encode the transition information given by the functions h_n into the advice tape.

Enumerate all functions $Q \times \Sigma \rightarrow Q$ by $\langle f_i \rangle_{i=1}^N$, where $N = (2k)^{2k|\Sigma|}$, and let $\Gamma = \langle c_i \rangle_{i=1}^N$ be the advice alphabet. Each character codes a possible transition behavior. For each $n \in \mathbb{N}$, pick a function f_i which respects h_n in the sense that if a class $C \in \mathcal{C}_n$ is associated to the state j , then for any character $a \in \Sigma$, $f_i(j, a)$ is the state associated to $h_n(C, a)$. Since not every state is associated to a class, f_i may behave arbitrarily on some inputs. Set the n^{th} character of the advice tape A to be c_i .

Finally, we define the transition function $\delta : Q \times \Gamma \times \Sigma \rightarrow Q$ by $\delta(j, c_i, a) = f_i(j, a)$. It is easy to check by induction on length that M accepts the string x if and only if $x \in L$. \square

3 On the role of alphabet size

In our proof of the Myhill-Nerode theorem with advice, we made use of a large advice alphabet. This raises the question of whether the size of the advice alphabet is essential.

Does there exist k such that if L is regular with advice, then already L is regular with some advice over an alphabet of size k ?

The answer is ‘no’. For $k \in \mathbb{N}$ let REGA_k be the set of languages that are regular in some advice with advice alphabet of size k . Then REGA_1 are the regular languages, and $\text{REGA}_k \subseteq \text{REGA}_{k+1}$. We prove that $\text{REGA}_k \neq \text{REGA}_{k+1}$ for all $k \in \mathbb{N}$.

For $A \in \{0, \dots, k\}^\omega$, let $\text{Pref}(A)$ be the language consisting of all prefixes (initial segments) of A . The set $\text{Pref}(A)$ is clearly regular with advice A , but if we choose A carefully, then $\text{Pref}(A)$ is not regular with any advice $B \in \{0, \dots, k-1\}^\omega$.

To simplify the proof, we will change the question to one about deterministic transducers. A *deterministic transducer* is a machine M which reads an infinite input string B and produces an infinite output string A . We denote this by $M[B] = A$. At each stage, M produces an output character based on an input character and its current state.

If we have an automaton M recognizing $\text{Pref}(A)$ with advice B , we can transform it into a deterministic transducer M^* such that $M^*[B] = A$. Note that on the run of M on the (infinite) input string A , M is always in an accepting state, and for each state and advice character pair (q, b) occurring in the run, there is exactly one input character a (the next character of A) such that $\delta(q, b, a)$ is an accepting state.

Without changing the language accepted by M , we may adjust the transition function so that for each pair of an accepting state $q \in F$ and an advice character b (even those pairs not appearing in the run of M on A), there is exactly one input character a such that $\delta(q, b, a)$ is an accepting state.

Now we define the deterministic transducer M^* so that it reads the advice string and at each stage produces the unique acceptable input character as output. That is, M^* has the same state set as M , and if M^* reads the advice character b in state q , it produces the output character a as above and transitions to $\delta(q, b, a)$.

Proposition 5. *There exists an infinite string A over alphabet $\Sigma = \{0, \dots, k\}$ such that for every deterministic transducer M (with input alphabet $\Gamma = \{0, \dots, k-1\}$ and output alphabet Σ) and every infinite string B over Γ , $M[B] \neq A$.*

Proof. We diagonalize. For each deterministic transducer M with input alphabet Γ and output alphabet Σ , we will produce a finite string u_M from Σ such that u_M does not appear as a substring of $M[B]$ for any string $B \in \Gamma^\omega$.

Let Q be the states of M . For $q \in Q$ and $b \in \Gamma$, write $M^q[b]$ for the character in Σ produced by M in state q upon reading the character b . Among the $k|Q|$ characters $M^q[b]$ (parametrized by $b \in \Gamma$ and $q \in Q$), there must be a character, say $a \in \Sigma$, that occurs at most $\frac{k}{k+1}|Q|$ times. Let a be the first character of u_M . Let Q' be those states $q' \in Q$ such that for some $b \in \Gamma$ and $q \in Q$, $M^q[b] = a$ and M transitions to q' . Note that $|Q'| \leq \frac{k}{k+1}|Q| < |Q|$, so Q' is a proper subset of Q . Replace Q by Q' and repeat to get the next character of u_M and a new set of states $Q'' \subset Q'$. After a finite number of steps, we will have $|Q^{(n)}| = 0$, at which point we have finished constructing u_M .

Now we claim that u_M cannot appear as a substring of $M[B]$ for any $B \in \Gamma^\omega$. Suppose for contradiction that it does appear in the run of M on B . At the step in which M begins producing u_M , it is in some state $q \in Q$. It reads a character from B , outputs the first character of u_M , and transitions into a state in Q' . After the next step, M transitions into a state in Q'' . After n steps, regardless of the content of B , M must be in a state in $Q^{(n)}$. But $Q^{(n)}$ is empty.

Concatenating the countably many strings u_M , we obtain our string A . For every deterministic transducer M and every infinite string B , $M[B] \neq A$, witnessed by the presence of u_M as a substring of A . \square

4 Nonterminating automata with advice

In the introduction we defined an automaton with advice as one that terminates after reading its finite input string (Definition 1). In this section, if L is regular with advice A we will say that L is *terminating regular with advice A* . There is another way to define automaton with advice, in what we call the non-terminating model. Here automata are defined with an infinitary acceptance condition. After the input string ends, the automaton continues to read the advice, producing an infinite run. The run is successful if it satisfies the acceptance condition. We work with deterministic Muller automata, in which the input string is accepted if the set of states visited infinitely often is in the collection F of accepting sets of states.

Definition 6. *A nonterminating automaton with advice is a tuple $M = (Q, \Sigma, \Gamma, A, \delta, q_0, F)$, with data just as before, except:*

- $\delta : Q \times \Gamma \times (\Sigma \cup \{\square\}) \rightarrow Q$ is the transition function. Here blank (\square) is a new symbol that is read once the input string has ended.
- $F \subseteq \mathcal{P}(Q)$ is the acceptance condition, where $\mathcal{P}(Q)$ is the powerset of Q .

The run of M on a string $w \in \Sigma^*$ is an infinite sequence of states $\alpha \in Q^\omega$ such that $\alpha_0 = q_0$ and:

- For $1 \leq n \leq |w|$: $\alpha_n = \delta(\alpha_{n-1}, A_n, w_n)$.
- For $n > |w|$: $\alpha_n = \delta(\alpha_{n-1}, A_n, \square)$.

We say a string $w \in \Sigma^*$ is accepted by an automaton with advice M if the set of states that appear infinitely often in α is an element of the acceptance condition F .

A language L is non-terminating regular with advice A if it is the language accepted by some non-terminating automaton with advice A . A language L is non-terminating regular with advice if there exists A such that L is non-terminating regular with advice A .

The distinction between the terminating and nonterminating models was irrelevant for our discussion of Myhill-Nerode because these models are equivalent if we do not fix the advice:

Proposition 7. For every language L , there exists advice A such that L is terminating regular with advice A if and only if there exists advice B such that L is non-terminating regular with advice B .

Proof. Given a terminating regular language, it is clearly also nonterminating regular with the same advice: the automaton can simply ignore the remainder of the advice, looping forever on the final state.

In the other direction, suppose $M = (Q, \Sigma, \Gamma, A, \delta, q_0, F)$ is a non-terminating automaton accepting L . Let α be the run of M on input w . Whether or not α is successful depends on whether or not M starting in state $\alpha_{|w|}$ and at position $n + 1$ of the advice tape accepts the empty string. This information can be encoded in the advice if we expand the advice alphabet. Formally, let B be the infinite string whose n^{th} letter is the pair (A_n, f_n) , where A_n is the n^{th} letter of the original advice A and $f_n \subseteq Q$ consists of all states q such that automaton $(Q, \Sigma, \Gamma, A[n+1, \infty), \delta, q, F)$ starting in state q and position $n + 1$ in the advice A accepts the empty string. Then L is terminating regular with advice A . \square

If $L_T(A)$ is the class of languages terminating regular with advice A and $L_N(A)$ is the class of languages nonterminating-regular with advice A , then $L_T(A) \subseteq L_N(A)$. However, the models are not equivalent for certain advice strings:

Proposition 8. There exists advice A such that $L_T(A) \neq L_N(A)$.

Proof. Let A be an infinite string on the binary alphabet $\Gamma = \{0, 1\}$ such that $\text{Pref}(A)$ is not regular (without advice). Let L be the language on the unary alphabet $\Sigma = \{0\}$ consisting of those strings of length n such that the $(n+1)^{st}$ character of A is a 1.

The language L is easily seen to be nonterminating regular with advice A . Indeed, let M be a machine which loops until it reads a blank. At that point, if the advice character is a 1, it transitions to an infinite loop at a state q with $\{q\} \in F$. If not, it transitions to an infinite loop at a state q' with $\{q'\} \notin F$.

But L is not terminating regular with advice A . The intuition is that when the input string ends, the automaton cannot guess the next advice character. Suppose for contradiction that $M = (Q, \Sigma, \Gamma, A, \delta, q_0, F)$ is a machine in the terminating model recognizing L with advice A . Then we can construct a machine $M' = (Q', \Gamma, \delta', q'_0, F')$ recognizing $\text{Pref}(A)$ (without advice).

Let $Q' = Q \cup \{r\}$, where r is a single new state, $F' = Q \subset Q'$, and $q'_0 = q_0$. Now define $\delta' : Q' \times \Gamma \rightarrow Q'$ as follows:

$$\delta'(q, a) = \begin{cases} \delta(q, a, 0) & \text{if } q \in F \text{ and } a = 1 \text{ or if } q \in Q \setminus F \text{ and } a = 0 \\ r & \text{if } q = r \text{ or if } q \in F \text{ and } a = 0 \text{ or if } q \in Q \setminus F \text{ and } a = 1 \end{cases}$$

If a state q was an accepting state of M , it expects the next character of A to be a 1, and if q was a rejecting state of M , it expects the next character of A to be a 0. Then M' recognizes $\text{Pref}(A)$, contradicting our assumption on the complexity of A . \square

5 Tree automata with advice

The classical Myhill-Nerode theorem has a natural generalization to (leaf-to-root deterministic) tree automata [6]. Our refinement can be easily adapted to this setting, giving a characterization of the sets of labeled trees which are regular with advice.

Definition 9. A (finite binary) tree $t \subset \{0, 1\}^*$ is a finite set of binary strings, called positions, which is downwards closed, in the sense that if $w \in t$ and v is an initial segment of w , then $v \in t$. A labeled tree is a pair (t, l) , where t is a tree and l is a function $t \rightarrow \Sigma$ for some finite alphabet Σ .

The root of a nonempty tree is the empty string λ . The children of a position $w \in t$ are the positions $w0$ and $w1$.

Given a tree t and a position $w \in t$ such that w has a child $w' \notin t$, we call the child w' a graft site. We also consider the empty string λ to be a graft site of the empty tree ε . If u is a graft site of t , then for any other tree x we define a new tree $t|_u x = t \cup \{uw \mid w \in x\}$. If t and x are labeled from Σ by l_t and l_x , then $t|_u x$ is also labeled from Σ : $l(w) = l_t(w)$ for $w \in t$ and $l(uw) = l_x(w)$ for $w \in x$.

An advice tree is a labeling of the complete binary tree $\{0, 1\}^*$ by a finite advice alphabet Γ , that is, a function $A : \{0, 1\}^* \rightarrow \Gamma$.

A tree automaton operates on a labeled tree t by assigning to all positions not in the tree a prescribed initial state. It then works inductively toward the root of the tree, assigning a state to each position in the tree based on the two states assigned to the children of that position and the label at that position. The automaton accepts if an accepting state is assigned to the root. A tree automaton with advice additionally has access to the advice character $A(u)$ when assigning a state to position $u \in t$.

Definition 10. A tree automaton with advice is a tuple $M = (Q, \Sigma, \Gamma, A, \delta, q_0, F)$.

1. Q is the finite set of states of the automaton.
2. Σ is a finite set of symbols called the input alphabet.
3. Γ is a finite set of symbols called the advice alphabet.
4. $A : \{0, 1\}^* \rightarrow \Gamma$ is the advice tree.
5. $\delta : Q \times Q \times \Gamma \times \Sigma \rightarrow Q$ is the transition function.
6. $q_0 \in Q$ is the initial state.
7. $F \subseteq Q$ is the acceptance condition.

The run of M on a labeled tree (t, l) is an assignment $r : \{0, 1\}^* \rightarrow Q$ of a state to each position in the complete binary tree such that if $w \notin t$, $r(w) = q_0$, and if $w \in t$, $r(w) = \delta(r(w0), r(w1), A(w), l(w))$. Since t is finite, there is a unique such assignment.

A labeled tree (t, l) is accepted by M if $r(\lambda) \in F$. A set of labeled trees T is tree regular with advice if it is the set accepted by some tree automaton with advice.

Remark 11. We have just introduced a terminating model. Of course there is also a non-terminating model (in this case the automaton is non-deterministic, starts at the root, and a run is successful if every infinite path satisfies a Muller condition). Just as in string case the two models are equivalent if we don't fix the advice (cf. Proposition 7) and not necessarily equivalent if we do fix the advice (cf. Proposition 8).

Let Σ be a finite alphabet and T a set of trees labeled from Σ . Define an equivalence relation \equiv_T on the set of all trees labeled from Σ by $x \equiv_T y$ if for any labeled tree t and any graft site u of t , it holds that $t|_u x \in T \iff t|_u y \in T$.

Theorem 12 (Myhill-Nerode theorem for trees). [6] A set of labeled trees T is tree regular if and only if \equiv_T has finitely many equivalence classes.

The definition must be modified in the presence of advice: Given a position $v \in \{0, 1\}^*$, define the equivalence relation $\equiv_{T,v}$ on the set of all labeled trees by $x \equiv_{T,v} y$ if for any labeled tree t such that v is a graft site of t , it holds that $t|_v x \in T \iff t|_v y \in T$.

Theorem 13 (Myhill-Nerode theorem for trees with advice). A set of labeled trees T is tree regular with advice if and only if there is some $k \in \mathbb{N}$ such that for all $v \in \{0, 1\}^*$, $\equiv_{T,v}$ has at most k equivalence classes.

The idea of the proof is the same as in the finite string case. If a set of labeled trees is regular with advice, the number of $\equiv_{T,v}$ -classes must be bounded by the number of states. Conversely, given a uniform bound k , we may construct an automaton by associating a state to each equivalence class and encoding the transition information into the advice tree.

We include the proof for completeness.

Proof. Suppose T is tree regular with advice. Let $M = (Q, \Sigma, \Gamma, A, \delta, q_0, F)$ be the tree automaton recognizing T , and let $k = |Q|$. Now assume that for some position v , $\equiv_{T,v}$ divides the set of labeled trees into n equivalence classes, with $n > k$. Pick representative labeled trees x_1, \dots, x_n in these classes. Let t be a labeled tree with graft site v , and let $q_i \in Q$ be the state associated to the position v (which is the root of x_i) in the run of M on $t|_v x_i$. Note that this is independent of the choice of t . Since $n > |Q|$, $q_i = q_j$ for some $i \neq j$. Then for any tree t with graft site v , M accepts $t|_v x_i$ if and only if M accepts $t|_v x_j$, since the states of M assigned to positions in the base tree t only depends on the states assigned to positions in the grafted tree based on which state is assigned to the string v . This contradicts the assumption that x_i and x_j are representatives of distinct equivalence classes.

Conversely, suppose we have such a bound k . For each position v , let \mathcal{C}_v be the collection of equivalence classes of $\equiv_{L,v}$, so $|\mathcal{C}_v| \leq k$. Note that \mathcal{C}_λ consists of at most two classes. Since the empty tree ϵ is the only tree with graft site λ , and $\epsilon|_\lambda t = t$, $s \equiv_{T,\lambda} t$ means that s and t are both in T or both not in T .

We must construct an automaton M recognizing T . Let $Q = \{1, \dots, k\}$ be the set of states, and let $F = \{1\} \subset Q$. Set $q_0 = 1$ if the empty tree is in T and $q_0 = 2$ otherwise. For each position v , we will associate one state to each equivalence class in \mathcal{C}_v . This can be done arbitrarily, except for two requirements:

- The state (1 or 2) named q_0 must always be associated to the equivalence class containing the empty tree.
- For \mathcal{C}_λ , the state 1 must be associated to the class consisting of those trees in T .

For each input character $a \in \Sigma$, we may form the singleton labeled tree consisting of just the root position labeled by a : $t_a = (\{\lambda\}, \lambda \rightarrow a)$. Now t_a has two graft sites, 0 and 1. If s and t are labeled trees, we can form the tree $t_{(s,a,t)} = (t_a|_0 s)|_1 t$. Now for any position v , if $s \equiv_{T,v0} s'$ and $t \equiv_{T,v1} t'$, then $t_{(s,a,t)} \equiv_{T,v} t_{(s',a,t')}$. Indeed, for any labeled tree x with graft site v ,

$$\begin{aligned} x|_v t_{(s,a,t)} \in T &\iff ((x|_v t_a)|_{v0} s)|_{v1} t \in T \\ &\iff ((x|_v t_a)|_{v0} s)|_{v1} t' \in T \\ &\iff ((x|_v t_a)|_{v1} t')|_{v0} s \in T \\ &\iff ((x|_v t_a)|_{v1} t')|_{v0} s' \in T \\ &\iff x|_v t_{(s',a,t')} \in T. \end{aligned}$$

This defines a function $h_v : \mathcal{C}_{v0} \times \mathcal{C}_{v1} \times \Sigma \rightarrow \mathcal{C}_v$ so that if $C \in \mathcal{C}_{v0}$, $C' \in \mathcal{C}_{v1}$ and $s \in C$, $t \in C'$, then $t_{(s,a,t)}$ is a member of the class $h_v(C, C', a)$. We now wish to encode this transition information into the advice tree.

Enumerate all functions $Q \times Q \times \Sigma \rightarrow Q$ by $\langle f_i \rangle_{i=1}^N$, where $N = k^{k^2|\Sigma|}$, and let $\Gamma = \langle c_i \rangle_{i=1}^N$ be the advice alphabet. Each character codes a possible transition behavior. For each position v , pick a function f_i which respects h_v in the sense that if states j and j' are associated to classes $C \in \mathcal{C}_{v0}$ and $C' \in \mathcal{C}_{v1}$, then for any character $a \in \Sigma$, $f_i(j, j', a)$ is the state associated to $h_v(C, C', a)$. Since not every state is associated to a class, f_i may behave arbitrarily on some inputs. Set $A(v) = c_i$.

Finally define the transition function $\delta : Q \times Q \times \Gamma \times \Sigma \rightarrow Q$ by $\delta(j, j', c_i, a) = f_i(j, j', a)$. It is easy to check that M accepts the labeled tree t if and only if $t \in T$. \square

References

- [1] V. Bárány (2006): *A Hierarchy of Automatic omega-Words having a Decidable MSO Theory*. In: *On-line proceedings of the 11th Journées Montoises*, Rennes 2006.
- [2] Olivier Carton & Wolfgang Thomas (2002): *The monadic theory of morphic infinite words and generalizations*. *Inf. Comput.* 176(1), pp. 51–65, doi:10.1006/inco.2001.3139.
- [3] T. Colcombet & C. Löding (2007): *Transforming structures by set interpretations*. *Logical Methods in Computer Science* 3(2), doi:10.2168/LMCS-3(2:4)2007.
- [4] C.C. Elgot & M.O. Rabin (1966): *Decidability of extensions of theory of successor*. *J. Symb. Log.* 31(2), pp. 169–181, doi:10.2307/2269808.
- [5] Séverine Fratani (2012): *Regular sets over extended tree structures*. *Theoretical Computer Science* 418(0), pp. 48 – 70, doi:10.1016/j.tcs.2011.10.020.
- [6] Dexter Kozen (1992): *On the Myhill-Nerode theorem for trees*. *Bull. Europ. Assoc. Theor. Comput. Sci.* 47, pp. 170–173.
- [7] André Nies (2007): *Describing groups*. *Bull. Symbolic Logic* 13(3), pp. 305–339, doi:10.2178/bsl/1186666149.
- [8] Alexander Moshe Rabinovich & Wolfgang Thomas (2006): *Decidable Theories of the Ordering of Natural Numbers with Unary Predicates*. In: *CSL*, pp. 562–574. Available at http://dx.doi.org/10.1007/11874683_37.
- [9] Sasha Rubin (2008): *Automata Presenting Structures: A Survey of the Finite String Case*. *Bulletin of Symbolic Logic* 14(2), pp. 169–209, doi:10.2178/bsl/1208442827.
- [10] Todor Tsankov (2011): *The additive group of the rationals does not have an automatic presentation*. *J. Symbolic Logic* 76(4), pp. 1341–1351, doi:10.2178/jsl/1318338853.

Interpretations in Trees with Countably Many Branches

Alexander Rabinovich

The Blavatnik School of Computer Science
Tel Aviv University
Tel Aviv, Israel
Email: rabinoa@post.tau.ac.il

Sasha Rubin

IST Austria and TU Vienna
Vienna, Austria

Abstract—We study the expressive power of logical interpretations on the class of *scattered* trees, namely those with countably many infinite branches. Scattered trees can be thought of as the tree analogue of scattered linear orders. Every scattered tree has an ordinal *rank* that reflects the structure of its infinite branches. We prove, roughly, that trees and orders of large rank cannot be interpreted in scattered trees of small rank. We consider a quite general notion of interpretation: each element of the interpreted structure is represented by a set of tuples of subsets of the interpreting tree. Our trees are countable, not necessarily finitely branching, and may have finitely many unary predicates as labellings. We also show how to replace injective set-interpretations in (not necessarily scattered) trees by ‘finitary’ set-interpretations.

Index Terms—Composition method, finite-set interpretations, infinite scattered trees, monadic second order logic.

I. INTRODUCTION

Monadic-second order logic (MSO) extends first-order logic with free variables that range over subsets of the domain, and allows quantification over them. When interpreted over trees MSO is expressive enough to capture interesting mathematics while still being manageable. Indeed, Rabin [20] proved that the MSO-theory of the full binary tree \mathfrak{T}_2 is decidable, and many other logical theories have been shown decidable by a reduction to this theory (see for instance the introductory sections in [20]). The *interpretation method* is a broad term that refers to effective reductions that are expressible logically, by a collection of formulas.¹

We consider interpretations that define, by MSO-formulas, structures \mathfrak{A} inside trees \mathfrak{T} . Our trees are subtrees of the countably-branching tree in the signature consisting of an order symbol (intended to be interpreted as the ancestor relation between nodes of the tree) and finitely many unary predicate symbols (intended to be interpreted as labellings of the nodes of the tree). In particular, nodes may have infinitely many children. A commonly occurring infinite tree is $(\mathbb{N}, <)$, also written ω . There are various kinds of interpretations depending on whether elements of the structure \mathfrak{A} are represented by nodes or (finite) sets of nodes of \mathfrak{T} . The latter are called (*finite*) *set-interpretations* and the former we

¹For instance, we learned in school that rational arithmetic is reducible to integer arithmetic by coding a rational by pairs of integers. In the terminology of this paper $(\mathbb{Q}, +, \times, =)$ is 2-dim point-interpretable in $(\mathbb{Z}, +, \times)$.

call *point-interpretations*² (Definitions 2.6 and 2.7). Moreover, each element of \mathfrak{A} is coded by at least one *tuple* of sets (the common size of the tuples is called the *dimension* of the interpretation); and if every element of \mathfrak{A} is coded by exactly one tuple of sets then the interpretation is called *injective*.

Why do interpretations in trees matter?

Interpretations allow one to transfer computational and logical properties from the interpreting structure to the interpreted structure.

Suppose that \mathfrak{A} is 1-dim point-interpretable in \mathfrak{T} . Then there is a uniform way to translate MSO-formulas in the signature of \mathfrak{A} to MSO-formulas in the signature of \mathfrak{T} (just replace atoms by their definitions, and relativize quantifiers). So the MSO-theory of \mathfrak{A} is computable in the MSO-theory of \mathfrak{T} . This explains the long term efforts to extend MSO decidability from ω and \mathfrak{T}_2 to their expansions by unary predicates [8], [10], [22], [26].

Suppose \mathfrak{A} is set-interpretable in \mathfrak{T} . Then there is a uniform way to translate FO-formulas in the signature of \mathfrak{A} to MSO-formulas in the signature of \mathfrak{T} , and in this case the FO-theory of \mathfrak{A} is computable in the MSO-theory of \mathfrak{T} . So the game here is to work out which structures \mathfrak{A} , known to have decidable FO-theory, are set-interpretable in trees with decidable MSO. Already Büchi noticed, in the language of automata, that the semigroup $(\mathbb{N}, +)$ is finite-set interpretable in ω . In modern terminology $(\mathbb{N}, +)$ is (finite-word) automatic. Also, the rational group $(\mathbb{Q}, +)$ is finite-set interpretable in a decidable expansion of ω (see [19]).

What is the expressive power of the interpretation method?

The research program that tries to outline the power of the interpretation method invariably has to prove non-interpretability results. Traditionally these were results about non-interpretability in expansions by unary predicates of linear orders (known as chains) [13], [18], [21]. However there are also non-interpretability results in graphs and trees. The Caucal hierarchy is a sequence $C_0, C_1 \dots$ of sets of graphs such that C_i is closed under 1-dim point-interpretations. There is a graph in C_i which is not 1-dim point-interpretable in any graph in C_j for $j < i$ [6].

²In the literature these are sometimes called MSO-interpretations.

A consequence of a general result in [7] is that if $\mathbb{P}_f(\mathfrak{A})$ is 1-dim finite-set weak-MSO interpretable in binary-branching \mathfrak{T} then already \mathfrak{A} is 1-dim weak-MSO point-interpretable in \mathfrak{T} .³ In the contrapositive this is a non-interpretability result; and indeed one of the main motivations in [7] is to reduce set-interpretability to the simpler point-interpretability. More recently, we learn that the real field $(\mathbb{R}, +, \times)$ is not set-interpretable in ω [1]. Looking at the proof we see that it goes through for any expansion of ω . Also, the rational group $(\mathbb{Q}, +)$ is not set-interpretable in ω (devoid of unary predicates) [27]. Much work in automatic structures is about proving that certain classes of structures are not (finite) set-interpretable in ω or \mathfrak{T}_2 (see [4], [15], [24]).

Overview

It is intuitively obvious that the more complex a tree the more it can interpret. We add weight to this contention by considering interpretations in trees with countably many infinite branches. We call these **scattered trees** since they are exactly the trees that do not embed the full binary tree. This name mimics the fact that linear orders that do not embed the rational order are called scattered orders, see [23]. The measure of complexity associates to every order and tree \mathfrak{A} an ordinal rank(\mathfrak{A}) (Definition 2.5). The rank of a tree reflects the structure of its infinite branches (similar to Cantor-Bendixson rank). All finite trees have rank 0, the line ω has rank 1, while any tree that embeds the full binary tree \mathfrak{T}_2 has rank ∞ (which is greater than all ordinals, and thus of maximum complexity). The rank of an order resembles Hausdorff ranks; thus ordinal ω^α has rank α . We prove, intuitively, that the rank of a tree \mathfrak{T} limits the possible ranks of orders and trees interpretable in \mathfrak{T} . Compare this with the fact that every countable order is point interpretable in some expansion of the non-scattered tree \mathfrak{T}_2 .

We emphasize that our results are of the form ‘ \mathfrak{A} is not interpretable in any expansion of \mathfrak{T} by unary predicates’, whether or not the expanded tree is decidable or even finitely presented. This is in line with previous investigations of the expressive power of interpretations in expansion of orders (called chains) [13], [18], [21] and in expansions of trees [7].

Technical contribution and related work

Point interpretations in scattered trees: These are simpler than set-interpretations and so proofs will not appear here. We prove that if an order or tree \mathfrak{A} is 1-dim point-interpretable in a tree \mathfrak{T} then $\text{rank}(\mathfrak{A}) \leq \text{rank}(\mathfrak{T})$. An immediate consequence is that neither \mathbb{Q} nor \mathfrak{T}_2 is point-interpretable in any scattered tree. This is an analogue of the result that neither \mathbb{Q} nor \mathfrak{T}_2 is point-interpretable in any scattered chain [21][Lemma 2.2].

³The structure $\mathbb{P}(\mathfrak{A})$ expands $(2^A, \subset)$ by the relations of \mathfrak{A} on singleton sets. So $\mathbb{P}(\mathbb{Q})$ is $(2^\mathbb{Q}, \subset, <)$ where for $X, Y \in 2^\mathbb{Q}$, $X < Y$ if and only if $X = \{x\}$, $Y = \{y\}$ and the rational x is less than the rational y . The structure $\mathbb{P}_f(\mathfrak{A})$ is the substructure of $\mathbb{P}(\mathfrak{A})$ consisting of finite subsets of A . By weak-MSO interpretation we mean an interpretation in which additionally bound variables vary over finite sets.

Finite-set interpretations in expansions of ω : We prove that no scattered order or scattered tree of non-finite rank is finite-set interpretable in any expansion by unary predicates of ω (Section III). This generalizes an early breakthrough in the area of automatic structures that no scattered order or scattered tree of non-finite rank is finite-set interpretable in ω (devoid of any unary predicate) [9].

Finite-set interpretations in scattered trees: We prove that there is an ordinal function G such that no ordinal of rank $\geq G(\alpha)$ is finite-set interpretable in any scattered tree of rank $\leq \alpha$. We may take $G(n) = \omega^n$ for $0 \leq n < \omega$ and $G(\alpha) = \omega^{\alpha+1}$ for $\alpha \geq \omega$ (Theorem 4.1).

Injective set-interpretations in scattered trees: We prove that no ordinal of rank $\geq G(\alpha)$ is injectively set-interpretable in any scattered tree of rank $\leq \alpha$ (Corollary 5.3). So neither $\mathbb{P}(\mathbb{Q})$ nor $\mathbb{P}(\mathfrak{T}_2)$ are injectively set-interpretable in any scattered tree (Corollary 5.4). Compare this with the fact that $\mathbb{P}(\mathbb{Q})$ and $\mathbb{P}(\mathfrak{T}_2)$ are injectively set-interpretable in \mathfrak{T}_2 . We conjecture that neither $\mathbb{P}(\mathbb{Q})$ nor $\mathbb{P}(\mathfrak{T}_2)$ are set-interpretable in any scattered tree.

Finitary-set interpretations in arbitrary trees: The previous result about injective set-interpretations follows from a theorem that is of independent interest. Even though set-interpretations allow one to interpret uncountable structures \mathfrak{A} , we do not study these. Instead consider the following general question: if \mathfrak{A} is countable and set-interpretable in (not necessarily scattered) tree \mathfrak{T} is \mathfrak{A} finite-set interpretable in \mathfrak{T} ? We do not solve this difficult problem. We establish a result of the same principle: if \mathfrak{A} is countable and injectively set-interpretable in tree \mathfrak{T} then there is an injective set interpretation of \mathfrak{A} in \mathfrak{T} for which the domain consists of tuples of finite sets and unlabelled-trees with finitely many infinite branches. Thus we manage to replace injective set-interpretations by these injective ‘finitary’-set interpretations. Similar ideas also give: if \mathfrak{A} is countable and injectively set-interpretable in a scattered tree \mathfrak{T} of *finite rank* then \mathfrak{A} is finite-set interpretable in \mathfrak{T} . We do not know if this holds for $\text{rank}(\mathfrak{T}) = \omega$.

Hierarchy strictness: For x a type of interpretation define \mathcal{I}_α^x as the set of structures that are x -interpretable in labelled trees of rank $\leq \alpha$. Clearly if $\alpha < \beta$ then $\mathcal{I}_\alpha^x \subseteq \mathcal{I}_\beta^x$. We have proven that these sets can be separated by ordinals; moreover, the bounds are tight. In summary, the hierarchies of injective set-interpretations, finite-set interpretations, and 1-dim point interpretations are strict: if $\alpha < \beta$ then $\mathcal{I}_\alpha^x \subsetneq \mathcal{I}_\beta^x$.

A note about technicalities. As far as the objects of study are concerned, the reader should have passing familiarity with linear orders and ordinal arithmetic (see [23]) and logical interpretations (see [14]). The central proof tool is Shelah’s composition theorem (see [11] for a readable account). Ideas from the proof in Section III are used in Section IV. Section V can be read independently.

II. DEFINITIONS AND PRELIMINARIES

The structures in this paper have finite relational signatures Δ , typically of the form $\Delta_l := \{\prec, P_1, \dots, P_l\}$ where \prec is

a binary predicate symbol and each P_i is a unary predicate symbol. In trees \prec represents the ancestor relation and in orders \prec represents element comparison. The tuple \overline{P} represents a labelling of the domain by elements of $\{0, 1\}^l$. In the next sections we define labelled linear orders (called chains) and labelled trees. Informally, the trees in this paper are subtrees of the countably-branching infinite tree of height ω , with unordered siblings, and expanded by finitely many unary predicates. The operations on these objects (sums) allow us to define the scattered orders and trees. Countable means finite or countably infinite. If unspecified structures in this paper are countable. We reserve $<$ for the ordering on ordinals. We write ω for the smallest infinite ordinal. The domain of a structure named \mathfrak{A} is written A . The expansion of \mathfrak{A} by predicates \overline{V} is written $(\mathfrak{A}, \overline{V})$. If $B \subset A$ then write (B, \overline{V}) for the substructure of $(\mathfrak{A}, \overline{V})$ on domain B .

A. Labelled Orders

An l -chain is a labelled linear order $\mathfrak{L} = (L, \prec, P_1, \dots, P_l)$. If $l = 0$ we talk about a linear order, or just order.

Definition 2.1 (Sums of l -chains): Given order $(\text{Ind}, \prec_{\text{ind}})$ and for every $i \in \text{Ind}$ an l -chain $\mathfrak{L}_i = (L_i, \prec_i, P_{i1}, \dots, P_{il})$ the sum $\sum_{\text{Ind}} \mathfrak{L}_i$ is defined as the l -chain with domain $\bigcup_{i \in \text{Ind}} \{i\} \times L_i$, ordering \prec defined by $(i, a) \prec (i', a')$ if and only if $i \prec_{\text{ind}} i'$ or $(i = i'$ and $a \prec_i a')$, and the k th unary predicate defined by $\bigcup_{i \in \text{Ind}} \{i\} \times P_{ik}$.

Write ω for the order type of the positive integers with the usual order, ω^* for that of the negative integers, and \mathbf{n} for the order type of the first n positive integers.

Definition 2.2 (scattered orders and ranks): Define sets of orders \mathfrak{B}_α and \mathfrak{L}_α by transfinite induction.

- $\mathfrak{B}_0 := \{\mathbf{1}\}$.
- \mathfrak{L}_α consists of $\sum_{\text{Ind}} \mathfrak{L}_i$ where Ind is finite and $\mathfrak{L}_i \in \mathfrak{B}_\alpha$.
- \mathfrak{B}_α consists of $\sum_{\text{Ind}} \mathfrak{L}_i$ where Ind has order type ω or ω^* and for all $i \in \text{Ind}$, $\mathfrak{L}_i \in \bigcup_{\beta < \alpha} \mathfrak{L}_\beta$.

An order \mathfrak{L} is *scattered* if it is in \mathfrak{L}_α for some α and the minimal such α is called the *rank* of \mathfrak{L} , written $\text{rank}(\mathfrak{L})$. The rank of a non-scattered order, written ∞ , is defined to be greater than all countable ordinals.⁴ The rank of a chain is defined as the rank of its underlying linear order.

The rank of a countable scattered order is countable and the rank of the ordinal written in Cantor-normal form $\sum_{i \leq m} \omega^{\alpha_i}$ is α_1 . In particular ω^α is the least ordinal of rank α . The following pigeonhole principle for linear orders is used so often that we isolate it here.

Lemma 2.3 (partition property for orders): If the domain of an order \mathfrak{L} is partitioned into finitely many pieces, then the order on at least one of the pieces has the same rank as that of \mathfrak{L} . If \mathfrak{L} has order type ω^α then at least one of the pieces has order type ω^α .

B. Labelled Trees

An l -labelled tree (or l -tree) is a structure

$$\mathfrak{T} = (T, \prec, P_1, \dots, P_l)$$

⁴Non-scattered orders can also be given an ordinal rank (see [23]) though we do not need this notion.

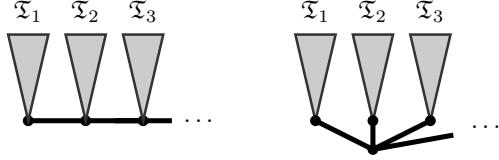


Fig. 1. An ω -sum (left), and ω -glueing.

where each $P_i \subseteq T$ and

- T is non-empty, partially ordered by \prec with unique minimal element (the *root* r);
- every $\{y \in T \mid y \preceq x\}$ is a finite linear order.

Terminology. If $l = 0$ the tree is *unlabelled*. A node v is a *child* of a node u if $u \prec v$ and there is no z with $u \prec z \prec v$. If every node has finitely many children the tree is *finitely branching*; otherwise it is *countably-branching*. If b is the smallest integer with the property that every node has at most b children then the tree is *b-ary branching*. If for non-empty $T' \subset T$ the substructure $\mathfrak{T} \upharpoonright T'$ is also a tree (ie. has a unique minimal element) then \mathfrak{T}' is a *subtree* of \mathfrak{T} . A typical example of a subtree is the *subtree of \mathfrak{T} rooted at $x \in T$* , written $\mathfrak{T}_{\geq x}$ and defined by the domain $\{u \in T \mid u \succeq x\}$. Another example is given by a *downward-closed set* $I \subset T$ (ie. $t \prec i \in I \implies t \in I$). A subset X is a *branch* if it is linearly-ordered by \prec and maximal with respect to set inclusion. Branches are subtrees and may be finite or infinite.

Definition 2.4 (tree sum): Given an unlabelled tree $(\text{Ind}, \prec_{\text{ind}})$ and for every $i \in \text{Ind}$ an l -tree $\mathfrak{T}_i = (T_i, \prec_i, P_{i1}, \dots, P_{il})$ with root r_i , the *sum* $\sum_{\text{Ind}} \mathfrak{T}_i$ is the l -tree with domain $\bigcup_{i \in \text{Ind}} \{i\} \times T_i$; ancestor relation defined by $(i, a) \prec (i', a')$ if and only if $(i = i'$ and $a \prec_i a')$ or $(i \prec_{\text{ind}} i'$ and $a = r_i)$; and the k th unary predicate P_k defined by $\bigcup_{i \in \text{Ind}} \{i\} \times P_{ik}$.

Terminology. Suppose $\mathfrak{T} = \sum_{\text{Ind}} \mathfrak{T}_i$. If Ind is finite then \mathfrak{T} is a *finite sum of \mathfrak{T}_i s*; if $(\text{Ind}, \prec_{\text{ind}})$ is a linear order of type ω then $\sum_{\text{Ind}} \mathfrak{T}_i$ is an ω -sum of \mathfrak{T}_i s. If $(\text{Ind}, \prec_{\text{ind}})$ consists of a root r and $n \leq \omega$ children, and \mathfrak{T}_r is a singleton tree, then \mathfrak{T} is an (n) -glueing of \mathfrak{T}_i s.

We now define ranks and scattered trees. The idea is that ω -sums and ω -glueings increase the rank while finite sums and finite glueings do not.

Definition 2.5 (scattered trees and ranks): Define families of unlabelled trees \mathfrak{F}_α and \mathfrak{G}_α by transfinite induction.

- \mathfrak{F}_0 consists of the unique tree with a single element.
- \mathfrak{G}_α consists of finite sums of \mathfrak{T}_i s with $\mathfrak{T}_i \in \mathfrak{F}_\alpha$.
- \mathfrak{G}_α consists of ω -sums and ω -glueings of \mathfrak{T}_i s where $\mathfrak{T}_i \in \bigcup_{\beta < \alpha} \mathfrak{G}_\beta$.

A tree \mathfrak{T} is *scattered* if it is in \mathfrak{G}_α for some α and the minimal such α is called the *rank* of \mathfrak{T} . The rank of a non-scattered tree, written ∞ , is defined to be greater than all ordinals. The rank of a labelled tree is defined as the rank of the unlabelled tree formed by removing the labels.

Thus the finite trees all have rank 0; the trees of rank 1 are ω -sums or ω -glueings of finite trees, and finite sums of these.

An *embedding* of unlabelled tree \mathfrak{T} in unlabelled tree \mathfrak{T}' is an injective function $f : T \rightarrow T'$ such that $x \prec y \implies f(x) \prec f(y)$. The following lore clarifies the status of scattered trees: tree \mathfrak{T} is scattered if and only if the complete infinite binary tree does not embed in \mathfrak{T} if and only if \mathfrak{T} has countably many infinite branches.

C. MSO and Interpretations

Monadic second order (MSO) logic consists of *MSO-variables* X, Y, Z, \dots that are interpreted as subsets of the domain, and allows quantification over these variables. The non-logical symbols are those from signature Δ and the binary predicate symbol \subseteq (representing set containment). Atomic formulas are those using symbols from Δ as well as those such as $X \subseteq Y$. Formulas are built from atomic formulas by applying Boolean connectives and universal and existential quantification of variables. We use abbreviations such as $X = Y, X \subset Y$ and $X \cap Y = \emptyset$. For convenience we may use FO-variables x, y, z, \dots since ‘ X is a singleton’ is definable in MSO.

A Δ -structure \mathfrak{A} consists of a domain A and for each k -ary predicate symbol R from Δ a k -ary predicate $R^{\mathfrak{A}} \subseteq A^k$. We may drop the superscript in $R^{\mathfrak{A}}$ when we are only dealing with one structure. An *expansion* of Δ -structure \mathfrak{A} by l many predicates is a structure $(\mathfrak{A}, P_1^{\mathfrak{A}}, \dots, P_l^{\mathfrak{A}})$ in the signature $\Delta \cup \{P_1, \dots, P_l\}$ where P_i are new unary predicate symbols. A Δ -formula is an MSO-formula in the signature Δ . Write $\mathbb{P}(A)$ for the set of all subsets of A . If $\varphi(X_1, \dots, X_m)$ is a Δ -formula and \mathfrak{A} is a Δ -structure then define

$$\varphi_{\mathfrak{A}} := \{(S_1, \dots, S_m) \in \mathbb{P}(A)^m \mid \mathfrak{A} \models \varphi(S_1, \dots, S_m)\}.$$

What types of interpretations do we consider?

The most well studied interpretations in this field are point-interpretations — the interpreting formulas are MSO (bound variables vary over sets) but their free variables are, effectively, first order variables. We also consider a more general notion called (*finite-*)set interpretations — here the interpreting formulas have free monadic variables that vary over (finite) sets; bound variables vary over arbitrary subsets. So in (*finite-*) set interpretations elements are coded by (finite) sets. Moreover interpretations may be multi-dim (elements are coded by tuples of finite sets) and not necessarily injective (each element can be coded by more than one tuple).

Definition 2.6 (set interpretation): A (d -dim) set interpretation Γ (in the signature Δ) consists of Δ -formulas

$$\partial_{\Gamma}(\overline{X_1}), \text{EQ}_{\Gamma}(\overline{X_1}, \overline{X_2}), \varphi_{\Gamma}^1(\overline{X_1}, \dots, \overline{X_{r_1}}), \dots, \varphi_{\Gamma}^k(\overline{X_1}, \dots, \overline{X_{r_k}})$$

where k is an integer, \overline{X}_i is a d -tuple of MSO-variables and in each formula all the variables named are distinct. Let \mathfrak{T} be a Δ -structure. If $\text{EQ}_{\Gamma}\mathfrak{T}$ is a congruence on the structure $(\partial_{\Gamma}\mathfrak{T}, \varphi_{\Gamma}^1\mathfrak{T}, \dots, \varphi_{\Gamma}^k\mathfrak{T})$ then define

$$\Gamma\mathfrak{T} := (\partial_{\Gamma}\mathfrak{T}, \varphi_{\Gamma}^1\mathfrak{T}, \dots, \varphi_{\Gamma}^k\mathfrak{T})_{/\text{EQ}_{\Gamma}\mathfrak{T}},$$

and say that $\Gamma\mathfrak{T}$, and any structure isomorphic to it, is *set-interpretable* in \mathfrak{T} via Γ . If $\text{EQ}_{\Gamma}(\overline{X}, \overline{Y})$ is the formula

$\bigwedge_{i < d} X_i = Y_i$ then Γ is an *injective* set-interpretation and $\Gamma\mathfrak{T}$ is *injectively* set-interpretable in \mathfrak{T} via Γ . If $d = 1$ then we might stress that Γ is a *1-dim set interpretation*.

We consider two particular types of interpretations depending, loosely speaking, on whether the free variables are taken to vary over elements of \mathfrak{T} or finite subsets of \mathfrak{T} .

Definition 2.7 (point- and finite-set interpretations): Let Γ be a set interpretation. If in Definition 2.6 one replaces $\partial_{\Gamma}\mathfrak{T}$ everywhere it occurs by $\partial_{\Gamma}\mathfrak{T}$ restricted to d -tuples of finite sets (resp. singletons) then we say that $\Gamma\mathfrak{T}$ is (d -dim) *finite-set interpretable* (resp. *point-interpretable*) in \mathfrak{T} .

Remark 2.8: Note that because finiteness is not definable in our trees, \mathfrak{A} may be finite-set interpretable in tree \mathfrak{T} while not being set interpretable in \mathfrak{T} .

Example 2.9: For every $n < \omega$ there is an n -dim injective point-interpretation of ω^n in ω . For instance, $\delta_{\Gamma}(X_1, \dots, X_n)$ states that each X_i is a singleton $\{x_i\}$; and $<_{\Gamma}(\overline{X}, \overline{Y})$ states that the least $i \leq n$ such that $x_i \neq y_i$ satisfies that $x_i < y_i$. Also, ω^n is injectively finite-set interpretable in ω . For instance, take $\delta_{\Gamma}(X)$ to be all sets of size n and order $X < Y$ if the smallest integer in the symmetric difference of X and Y is in X . The ordinal ω^ω is the least ordinal that is not finite-set interpretable in ω [9]. A corollary of Theorem 3.1 is that ω^ω is the least ordinal that is not finite-set interpretable in any expansion of ω by unary predicates.

Definition 2.10 (interpretation with parameters): Let Γ be an interpretation in the signature Δ such that every formula in Γ contains an additional m -tuple of free variables. Then Γ is called an *interpretation with m parameters*. Let \overline{S} be an m -tuple of subsets of a tree \mathfrak{T} . Then $\Gamma(\mathfrak{T}, \overline{S})$, and any structure isomorphic to it, is said to be *interpretable in \mathfrak{T} with m parameters via Γ* . A family $\{\mathfrak{B}_i\}$ is *interpretable in \mathfrak{T} with m parameters via Γ* if for every member \mathfrak{B}_i there exists an m -tuple \overline{S} such that \mathfrak{B}_i is isomorphic to $\Gamma(\mathfrak{T}, \overline{S})$.

Example 2.11: [20] The rational order \mathbb{Q} is injectively 1-dim point-interpretable with one parameter in the full binary tree $\mathfrak{T}_2 := (\{0, 1\}^*, \prec_{\text{pref}})$. The parameter may be taken as $R := \{0, 1\}^*1$ and allows one to distinguish left and right children and so define the lexicographic ordering on $\{0, 1\}^*$. Consequently, since every countable order embeds in the rational order, there is a 1-dim point-interpretation Γ such that the family of countable linear orders is injectively interpretable in \mathfrak{T}_2 with two parameters; one parameter is R and the other picks out the domain of the countable linear order.

The next proposition follows from the standard interpretation of the full countably-branching tree \mathfrak{T}_ω in \mathfrak{T}_2 [20].

Proposition 2.12: Every tree of rank α is 1-dim point-interpretable in a binary tree of rank α .

D. Composition Theorem for Tree-sums and Order-sums

Write Δ_l for the signature of order \prec with l unary predicate P_1, \dots, P_l symbols. Thus a Δ_l -structure \mathfrak{A} has the form $(A, \prec, P_1^{\mathfrak{A}}, \dots, P_l^{\mathfrak{A}})$. The quantifier rank of a formula φ , denoted $\text{qr}(\varphi)$, is the maximum depth of nesting of quantifiers in φ . For $r, l \in \mathbb{N}$ we denote by \mathfrak{Form}_l^r the set of formulas of

quantifier rank $\leq r$ and with free variables among X_1, \dots, X_l in signature $\{\prec\}$. For Δ_l -structures $\mathfrak{A}, \mathfrak{B}$ write $\mathfrak{A} \equiv_l^r \mathfrak{B}$ if for every $\varphi \in \mathfrak{Form}_l^r$,

$$\mathfrak{A} \models \varphi(P_1^{\mathfrak{A}}, \dots, P_l^{\mathfrak{A}}) \text{ if and only if } \mathfrak{B} \models \varphi(P_1^{\mathfrak{B}}, \dots, P_l^{\mathfrak{B}}).$$

Clearly \equiv_l^r is an equivalence relation and the set \mathfrak{Form}_l^r is infinite. Since the signature Δ_l is finite and relational the set \mathfrak{Form}_l^r contains only finitely many semantically distinct formulas so there are only finitely many \equiv_l^r -classes of Δ_l -structures. The following lemma isolates maximally consistent formulas.

Lemma 2.13 (Hintikka lemma): For $r, l \in \mathbb{N}$, there is a finite set $H_l^r \subseteq \mathfrak{Form}_l^r$ such that:

- 1) For every Δ_l -structure \mathfrak{A} there is a unique $\tau \in H_l^r$ with $\mathfrak{A} \models \tau(P_1^{\mathfrak{A}}, \dots, P_l^{\mathfrak{A}})$.
- 2) If $\tau \in H_l^r$ and $\varphi \in \mathfrak{Form}_l^r$, then either $\tau \models \varphi$ or $\tau \models \neg\varphi$.⁵

Elements of H_l^r are called (r, l) -Hintikka formulas. For every r, l we fix an enumeration $\tau_1(\overline{X}), \dots, \tau_{|H_l^r|}(\overline{X})$ of H_l^r .

Definition 2.14 (type of a structure): For Δ_l -structure \mathfrak{A} write $\text{Tp}_l^r(\mathfrak{A})$ for the unique $\tau(X_1, \dots, X_l) \in H_l^r$ such that $\mathfrak{A} \models \tau(P_1^{\mathfrak{A}}, \dots, P_l^{\mathfrak{A}})$, and call it the (r, l) -type of \mathfrak{A} .

Thus $\text{Tp}_l^r(\mathfrak{A})$ effectively determines for which formulas $\varphi \in \mathfrak{Form}_l^r$ it holds that $\mathfrak{A} \models \varphi(P_1^{\mathfrak{A}}, \dots, P_l^{\mathfrak{A}})$. Since l is often clear we may drop it and write $\text{Tp}^r(\mathfrak{A})$ and r -type.

We now discuss increasingly informative versions of the composition theorem for MSO over tree-sums and order-sums, see [25] or [11], [12] for details. The first, lets call it *weak composition*, says that the (r, l) -type of a sum depends on the (r, l) -types of its summands.

Theorem 2.15 (weak composition for tree- and order-sums): For every Ind , if $\text{Tp}_l^r(\mathfrak{A}_i) = \text{Tp}_l^r(\mathfrak{A}'_i)$ for all $i \in \text{Ind}$ then $\text{Tp}_l^r(\sum_{\text{Ind}} \mathfrak{A}_i) = \text{Tp}_l^r(\sum_{\text{Ind}} \mathfrak{A}'_i)$.

We use the following consequence of the weak composition for orders: for every r, l there is associative binary operation $+$ on the set H_l^r such that for all l -orders $\mathfrak{L}_1, \mathfrak{L}_2$ the formula $\text{Tp}_l^r(\mathfrak{L}_1) + \text{Tp}_l^r(\mathfrak{L}_2)$ is identical with the (r, l) -type of their sum $\text{Tp}_l^r(\sum \mathfrak{L}_i)$.

Definition 2.16 (partition of Ind): Let $\{\mathfrak{A}_i\}_{i \in \text{Ind}}$ be a family of l -structures. The H_l^r -partition (of Ind) induced by $\{\mathfrak{A}_i\}_{i \in \text{Ind}}$ is the $|H_l^r|$ -tuple \overline{Q} where $Q_k := \{i \in \text{Ind} \mid \text{Tp}_l^r(\mathfrak{A}_i) = \tau_k\}$ and τ_k is the k th formula in the enumeration of H_l^r . The H_l^r -expansion of $(\text{Ind}, \prec_{\text{Ind}})$ induced by $\{\mathfrak{A}_i\}_{i \in \text{Ind}}$ is the structure $(\text{Ind}, \prec_{\text{Ind}}, \overline{Q})$.

With this notation weak composition states that $\text{Tp}_l^r(\sum_{\text{Ind}} \mathfrak{A}_i)$ is determined by the H_l^r -expansion induced by $\{\mathfrak{A}_i\}_{\text{Ind}}$. It turns out that $\text{Tp}_l^r(\sum_{\text{Ind}} \mathfrak{A}_i)$ is already determined by some q -type of the H_l^r -expansion. In the next version q does not even depend on Ind :

Theorem 2.17 (composition for tree-sums): For every formula $\varphi \in \mathfrak{Form}_l^r$ there exists a formula $\theta(Y_1, \dots, Y_{|H_l^r|})$ ⁶ such

⁵Furthermore, H_l^r is computable from r, l , and there is an algorithm that given τ and φ decides between $\tau \models \varphi$ and $\tau \models \neg\varphi$. We do not use these facts.

⁶Moreover, θ is computable from φ , although we do not use this fact.

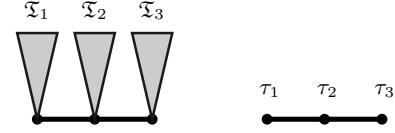


Fig. 2. Illustration of composition: The r -type of the tree (left) is determined by the q -type of the chain (right) where $\tau_i := \text{Tp}^r(\mathfrak{T}_i)$. This q -type is called a projected type.

that for every unlabelled tree $(\text{Ind}, \prec_{\text{Ind}})$ and family $\{\mathfrak{T}_i\}_{i \in \text{Ind}}$ of l -trees

$$\sum_{\text{Ind}} \mathfrak{T}_i \models \varphi \iff (\text{Ind}, \prec_{\text{Ind}}) \models \theta(\overline{Q})$$

where \overline{Q} is the H_l^r -partition induced by $\{\mathfrak{T}_i\}_{i \in \text{Ind}}$. The quantifier-rank of θ depends only on r and l and so is written $q(r, l)$.

Projected Types: The notions in the remainder of this section are only used in Section IV. We visualise (Figure 2) the r -type of \mathfrak{T}_i projected onto i and introduce notation to capture this. A projected (r, l) -Hintikka formula is a $(q(r, l), |H_l^r|)$ -Hintikka formula. The projected (r, l) -type of family $\{\mathfrak{T}_i\}_{i \in \text{Ind}}$ of l -trees is the $(q(r, l), |H_l^r|)$ -type of the H_l^r -expansion of $(\text{Ind}, \prec_{\text{Ind}})$ induced by $\{\mathfrak{T}_i\}_{\text{Ind}}$. Note that the projected type determines which quantifier-rank r formulas hold in $\sum_{\text{Ind}} \mathfrak{T}_i$.

When dealing with scattered trees Ind is often ω , so the type of an ω -sum of trees reduces to the type of an expansion of ω . Given a family $\{\mathfrak{T}_i\}_{i < n}$ of l -trees ($n \leq \omega$) and m -tuple \overline{A} write $\text{PrTp}(\overline{A})_{[i,j]}^r$ for the projected $(r, l+m)$ -type of family $\{(\mathfrak{T}_k, \overline{A})\}_{k \in [i,j]}$. Under this notation we have

$$\text{PrTp}(\overline{A})_{[0,n]}^r = \text{PrTp}(\overline{A})_{[0,a]}^r + \text{PrTp}(\overline{A})_{[a,n]}^r.$$

The following proposition says that if elements of $\Gamma \sum_{\omega} \mathfrak{T}_i$ are all in an ‘interval sum’ of \mathfrak{T} then $\Gamma \sum_{\omega} \mathfrak{T}_i$ is interpretable in this interval.

Proposition 2.18: Suppose $\mathfrak{T} = \sum_{\omega} \mathfrak{T}_i$ is an l -tree and Γ is d -dim set (resp. point-, finite-set) interpretation. If there exists $a < b$ such that $\mathfrak{T} \models \partial_{\Gamma}(W_1, \dots, W_d)$ implies $W_i \subseteq \sum_{i \in [a,b]} \mathfrak{T}_i$, then $\Gamma \mathfrak{T}$ is d -dim set (resp. point-, finite-set) interpretable in some expansion of $\sum_{i \in [a,b]} \mathfrak{T}_i$.

III. FINITE-SET INTERPRETATIONS IN EXPANSIONS OF ω

We write Δ_l for the signature consisting of \prec and l predicate symbols.

- Theorem 3.1:*
- 1) For every **finite-set** interpretation Γ in the signature Δ_l there exists an integer N_{Γ} such that for every expansion \mathcal{C} of ω by l unary predicates, if $\Gamma \mathcal{C}$ is a scattered order or scattered tree then its rank is at most N_{Γ} .
 - 2) In particular, no scattered order or scattered tree of non-finite rank is finite-set interpretable in any expansion of ω by unary predicates.

The second item immediately follows from the first. Moreover, the bound is tight since every ordinal of finite rank is finite-set (even many-dim point-) interpretable in ω . It was

shown in [9] that no ordinal of non-finite rank is finite-set interpretable in ω . That proof, which does not go through for expansions of ω , inspired Theorem 3.1.

From [3, Proposition 3.1] we can conclude that a countable structure that is set-interpretable with parameters in an expansion \mathcal{C} of ω is already finite-set interpretable with parameters in \mathcal{C} . Thus no countable scattered order or tree of non-finite rank is *set* interpretable in any expansion of ω .

The proof of Theorem 3.1 uses the following *rank properties* on a class of structures \mathcal{C} closed under isomorphism and under substructure:

- 1) Isomorphic structures in \mathcal{C} have the same rank.
- 2) If $\mathfrak{A} \in \mathcal{C}$ has rank α and A is partitioned into P_1, P_2 , then at least one of $\mathfrak{A} \upharpoonright P_1$ and $\mathfrak{A} \upharpoonright P_2$ has rank α .
- 3) There is a 1-dim point-interpretation Γ with point parameters such that for every \mathfrak{A} of finite rank k (infinite rank) there is a family of $k - 1$ (infinitely many) structures of distinct ranks interpretable with parameters in \mathfrak{A} via Γ .

Ranks on the class \mathcal{C} of linear orders (Definition 2.2) satisfy these properties. For the third property use the fact that if \mathfrak{L} has rank α then for every $\beta < \alpha$ there exist an open interval of \mathfrak{L} of rank β . Infact the domain formula $\partial_\Gamma(x, p_1, p_2)$ may be defined as $p_1 \prec x \prec p_2$ where the p_i s are parameters. Similarly the class \mathcal{C} of forests (ie. sets of trees) is closed under substructure (unlike the class of trees) and we may define a ranking on forests (agreeing with the ranking on trees) satisfying these three properties as follows: the rank of a set \mathfrak{S} of trees is the supremum of ranks of the trees in \mathfrak{S} . Theorem 3.1 is immediate from rank property 3) and the following.

Proposition 3.2: For every **set** interpretation Γ in the signature Δ_l with m parameters there exists an integer N_Γ such that if Γ interprets a family of scattered structures (orders or trees) $\{\mathcal{C}_i\}_{i \in I}$ with m -many **finite-set** parameters in some expansion \mathcal{C} of ω then the number of distinct ranks amongst the ranks of $\{\mathcal{C}_i\}_{i \in I}$ is at most N_Γ .

Proof: To help readability we prove the proposition for 1-dim interpretations and $m = 1$ (one parameter). However the same proof goes through for d -dim interpretations and m parameters — replace variables and parameters ranging over subsets of ω by those ranging over d -tuples of subsets of ω . We sometimes mention m and d in the proof below to help the reader generalize.

Let q be an upper bound on the quantifier-rank of the formulas in the interpretation Γ . Define N_Γ greater than the number of $(q, l + m + 2d)$ -Hintikka formulas, namely $|H_{l+m+2d}^q|$. Take a family $\{\mathcal{C}_n\}_{n \in I}$ of scattered orders of distinct ranks that is interpreted with m -many parameters in an expansion (ω, \bar{p}) via Γ . By assumption the m -many parameters are restricted to be finite subsets of ω .

Notation. For the rest of this proof we use lowercase p_i, w_i, \dots to refer to subsets of ω , and uppercase L_i, D_i, \dots to refer to sets of subsets of ω . For $z \subset \omega$ write $z[a, b)$ for $z \cap [a, b)$.

For every $n \in I$: fix a finite parameter $w_n \subset \omega$ so that $\Gamma(\omega, \bar{p}, w_n) \cong \mathcal{C}_n$; write D_n for the domain $\{x \mid (\omega, \bar{p}, w_n) \models \partial_\Gamma(x)\}$; write $=_n$ for the relation $\{(x, y) \mid (\omega, \bar{p}, w_n) \models$

$\text{EQ}_\Gamma(x, y)\}$; write \prec_n for the ordering $\{(x, y) \mid (\omega, \bar{p}, w_n) \models x \prec_\Gamma y\}$; and write \preceq_n for $\prec_n \cup =_n$.

For $z \subseteq \omega$ and $t \in \omega$ write $L_n(z, t)$ for the set of $x \in D_n$ such that $x[0, t) = z[0, t)$. That is, $L_n(z, t)$ consists of elements in the domain D_n that agree with z on the initial interval $[0, t)$. Note $L_n(z, 0) = D_n$ for all z .

Claim 1. For every index $n \in I$ there is $z_n \subseteq \omega$ such that for every $t \in \omega$ the rank of $(L_n(z_n, t), \prec_n)/=_n$ equals the rank of \mathcal{C}_n .

Proof of Claim 1. Fix n and suppose z_n has already been defined on the interval $[0, k)$ and for all $t \leq k$ the rank of $(L_n(z_n, t), \prec_n)/=_n$ equals the rank of \mathcal{C}_n . Partition the sets in $L_n(z_n, k)$ into two classes V_0, V_1 depending on whether or not k is in the set. By rank property 2) above at least one of $(V_i, \prec_n)/=_n$ has the same rank as \mathcal{C}_n ; say the class is represented by $\epsilon \in \{0, 1\}$. Put integer k in z_n if and only if $\epsilon = 1$. •

For every $n \in I$ fix z_n by Claim 1. Suppose, for a contradiction, there were more than N_Γ distinct ranks amongst the \mathcal{C}_i s. By Claim 1, and rank property 1) above, for every t there are more than N_Γ structures up to isomorphism amongst $(L_n(z_n, t), \prec_n)/=_n$ ($n \in I$). Pick finite $J_t \subset I$ of size greater than N_Γ indexing non-isomorphic structures (that is, $j \neq k \in J_t$ implies $(L_j(z_j, t), \prec_j)/=_j$ is not isomorphic to $(L_k(z_k, t), \prec_k)/=_k$). The following claim shows that the choice of N_Γ ensures that this is impossible.

Claim 2. There exists integer t and distinct indices $n, k \in J_t$ such that $(L_n(z_n, t), \prec_n)/=_n$ and $(L_k(z_k, t), \prec_k)/=_k$ are isomorphic.

Proof of Claim 2. Write $(\omega, \bar{v})[0, t)$ for the structure (ω, \bar{v}) restricted to domain $[0, t)$. By choice of N_Γ , for every t there exist distinct $n, k \in J_t$ such that $\text{Tp}^q(\omega, \bar{p}, w_n, z_n, z_n)[0, t) = \text{Tp}^q(\omega, \bar{p}, w_k, z_k, z_k)[0, t)$. Fix an integer t that is greater than all the integers in all the w_i s for $i \in J_t$; and take n, k as in the previous sentence. So $w_n, w_k \subset [0, t)$. For $x, y \subset [t, \omega)$,

$$\begin{aligned} \text{Tp}^q(\omega, \bar{p}, w_n, z_n[0, t) \cup x, z_n[0, t) \cup y) &= \\ \text{Tp}^q(\omega, \bar{p}, w_n, z_n, z_n)[0, t) + \text{Tp}^q(\omega, \bar{p}, \emptyset, x, y)[t, \omega) &= \\ \text{Tp}^q(\omega, \bar{p}, w_k, z_k, z_k)[0, t) + \text{Tp}^q(\omega, \bar{p}, \emptyset, x, y)[t, \omega) &= \\ \text{Tp}^q(\omega, \bar{p}, w_k, z_k[0, t) \cup x, z_k[0, t) \cup y). & \end{aligned}$$

Immediately then

- 1) $(z_n[0, t) \cup x) \in L_n(z_n, t)$ iff $(z_k[0, t) \cup x) \in L_k(z_k, t)$
- 2) $(z_n[0, t) \cup x) \prec_n (z_n[0, t) \cup y)$ iff $(z_k[0, t) \cup x) \prec_k (z_k[0, t) \cup y)$, and
- 3) $(z_n[0, t) \cup x) =_n (z_n[0, t) \cup y)$ iff $(z_k[0, t) \cup x) =_k (z_k[0, t) \cup y)$.

These properties ensure that the map $z_n[0, t) \cup x \xrightarrow{\phi} z_k[0, t) \cup x$ (where x ranges over subsets of $[t, \omega)$) induces an isomorphism $\Phi : (L_n(z_n, t), \prec_n)/=_n \rightarrow (L_k(z_k, t), \prec_k)/=_k$. Indeed Φ is a well-defined function by item 3); it is onto by item 1); and order-preserving by item 2). •

IV. FINITE-SET INTERPRETATIONS IN SCATTERED TREES

Theorem 4.1: There is an ordinal function G such that no ordinal of rank $\geq G(\alpha)$ is finite-set interpretable in any

labelled tree of rank $\leq \alpha$. We may take $G(n) = \omega^n$ for $0 \leq n < \omega$ and $G(\alpha) = \omega^{\alpha+1}$ for $\alpha \geq \omega$.

A. Natural Sum and Product on Ordinals

The proof has some similarities with that of Proposition 3.2 but requires additional machinery, including the use of the natural-sum \oplus (also called Hessenberg-sum) and natural-product \otimes (also called Hausdorff-product) on ordinals. These operations were introduced in [5] and can be thought of as addition and multiplication of polynomials in ω . The natural-sum is a commutative, associative binary operation \oplus on ordinals. Suppose $\alpha = \sum_{i < m} \omega^{\alpha_i}$ and $\beta = \sum_{j < n} \omega^{\beta_j}$ are in Cantor-normal-form. Then $\alpha \oplus \beta$ is defined as the sum (as in Definition 2.1) of all ω^{α_i} and ω^{β_j} arranged in non-increasing order. Similarly the natural-product is a commutative, associative binary operation \otimes on ordinals. Define $\alpha \otimes \beta$ as the natural sum of all $\omega^{\alpha_i \oplus \beta_j}$. We implicitly use easy properties of natural ordinal arithmetic: for instance, the natural sum or natural product of countable ordinals is countable; if $\alpha, \beta < \omega^\gamma$ then $\alpha \oplus \beta < \omega^\gamma$; if $\alpha, \beta < \omega^{\omega^\gamma}$ then $\alpha \otimes \beta < \omega^{\omega^\gamma}$. Here is a central property of \otimes . A function $f : (\alpha_1 \times \cdots \times \alpha_k) \rightarrow \gamma$ is *coordinate-wise non-decreasing* if for all $(\delta_1, \dots, \delta_k)$ in the domain of f and all $n \leq k$ and all ordinals δ with $\delta_n \leq \delta < \alpha_n$, it holds that $f(\delta_1, \dots, \delta_k) \leq f(\delta_1, \dots, \delta_{n-1}, \delta, \delta_{n+1}, \dots, \delta_k)$.

Lemma 4.2: [5] If $f : (\alpha_1 \times \cdots \times \alpha_k) \rightarrow \gamma$ is onto and coordinate-wise non-decreasing then $\gamma \leq \alpha_1 \otimes \cdots \otimes \alpha_k$.

B. Proof of Theorem 4.1

We illustrate the proof for dimension 1 to get a function G_1 and remark at the end how to deal with dimension d . It is enough to find a function G_1 such that if ω^δ is 1-dim finite set interpretable in a tree of rank α then $\delta < G(\alpha)$ (this is because $\omega^{\text{rank}(\beta)} \leq \beta$ and the ordinals that are finite-set interpretable in expansions of \mathfrak{T} are closed downwards). To this end, say ω^δ is finite-set interpretable in the l -tree \mathfrak{T} of rank α via Γ . Write D for the domain of the interpretation; \prec for $\prec_\Gamma \mathfrak{T}$, and $=_\Gamma$ for $\text{EQ}_\Gamma \mathfrak{T}$. Since $(D, \prec)/=_\Gamma$ is isomorphic to ω^δ , say via bijection f , for every ordinal $x < \omega^\delta$ pick a unique element in D from the equivalence class $f^{-1}(x)$ and call it the *code* of x . Let r be the largest quantifier-rank appearing in formulas of Γ . For the remainder write $\text{PrTp}(\overline{A})_{[i,j]}$ instead of $\text{PrTp}(\overline{A})_{[i,j]}^r$.

A note on the structure of the proof. We induct on α to bound δ . For the base case $G_1(0) := 1$ (since no infinite structure is interpretable in a finite one). For the remainder suppose $\alpha > 0$. We consider two cases: the first is that \mathfrak{T} is an ω -sum of \mathfrak{T}_i s of lower rank (these we call Case 1 trees); and the second is that \mathfrak{T} is a finite sum of Case 1 trees (called Case 2 trees). The case that \mathfrak{T} is an ω -glueing is reduced to these cases by Proposition 2.12.

Notation. For the rest of this proof we use lowercase p_i, w_i, \dots to refer to subsets of T , and uppercase L, R, \dots to refer to sets of subsets of T .

Case 1. Say \mathfrak{T} is an ω -sum of \mathfrak{T}_i s of lower rank. For interval $[a, b)$ write $T[a, b)$ for the set $\cup_{i \in [a, b)} T_i$.

The **aim** is to bound every $\beta < \delta$ in terms of r, l, d and $G_1(\alpha')$ (for $\alpha' < \alpha$). So take arbitrary $\beta < \delta$ and write $w \in D$ for the code of ω^β . By Lemma 2.3 (partition property for orders) for all t there exists projected $(r, l + 2d)$ -Hintikka formulas $\lambda_{w,t}$ and $\rho_{w,t}$ such that

$$D_{w,t} := \{x \in D \mid x \prec w \quad \text{and} \quad \text{PrTp}(w, x)_{[0,t)} = \lambda_{w,t} \quad \text{and} \quad \text{PrTp}(w, x)_{[t,\omega)} = \rho_{w,t}\}$$

ordered by \prec has order type ω^β . Define sets $L_{w,t}$ as

$$\{y \subset T[0, t) \mid y \text{ finite and } \text{PrTp}(w, y)_{[0,t)} = \lambda_{w,t}\}$$

and $R_{w,t} := \{z \subset T[t, \omega) \mid z \text{ finite and } \text{PrTp}(w, z)_{[t,\omega)} = \rho_{w,t}\}$.

In other words, every $x \in D_{w,t}$ satisfies that $x \cap [0, t) \in L_{w,t}$ and $x \cap [t, \omega) \in R_{w,t}$. Define binary relations $\prec_{L,w,t}$ and $=_{L,w,t}$ on $L_{w,t}$ by

$$\begin{aligned} y \prec_{L,w,t} y' &\text{ if } \exists z \in R_{w,t} \quad y \cup z \prec y' \cup z \\ y =_{L,w,t} y' &\text{ if } \exists z \in R_{w,t} \quad y \cup z =_\Gamma y' \cup z \end{aligned}$$

and binary relations $\prec_{R,w,t}$ and $=_{R,w,t}$ on $R_{w,t}$ by

$$\begin{aligned} y \prec_{R,w,t} y' &\text{ if } \exists z \in L_{w,t} \quad y \cup z \prec y' \cup z \\ y =_{R,w,t} y' &\text{ if } \exists z \in L_{w,t} \quad y \cup z =_\Gamma y' \cup z \end{aligned}$$

Let $\#w$ denote the smallest s such that $w \subset T[0, s]$. We will show that (\ddagger) for $t > \#w$ we can replace $\exists z$ by $\forall z$ in the above definitions.

Lemma A. For $t > \#w$, both $(L_{w,t}, \prec_{L,w,t})/=_L$ and $(R_{w,t}, \prec_{R,w,t})/=_L$ are well-orders.

We defer the proof of Lemma A. Write $\beta_{L,w,t}$ and $\beta_{R,w,t}$ for their respective order types and note (\star) that each is at most ω^β . The map $\Phi : \beta_{L,w,t} \times \beta_{R,w,t} \rightarrow \omega^\beta$ (induced by $(y, z) \mapsto y \cup z$) is surjective by definition of $L_{w,t}$ and $R_{w,t}$. It is co-ordinate wise non-decreasing by (\ddagger) . Apply Lemma 4.2 to conclude $(\star\star)$ that ordinal ω^β is at most $\beta_{L,w,t} \otimes \beta_{R,w,t}$.

The order $(L_{w,t}, \prec_{L,w,t})/=_L$ is finite-set interpretable in $\mathfrak{T}[0, t)$ (by Proposition 2.18). This tree has some rank $\alpha' < \alpha$, so apply induction and conclude $(\star\star\star)$ that $\beta_{L,w,t} < \omega^{G_1(\alpha')}$. All that is left is to bound $\beta_{R,w,t}$. For this we use a pigeonhole argument.

Lemma B. If w' codes $\omega^{\beta'} (\beta' < \delta)$ then for $t > \#w, \#w'$, if $\lambda_{w,t} = \lambda_{w',t}$ and $\rho_{w,t} = \rho_{w',t}$ then $\beta_{R,w,t} = \beta_{R,w',t}$.

We defer the proof of Lemma B. Define $\gamma(\alpha) := \sup_{\alpha' < \alpha} G_1(\alpha')$ and pick a sequence of ordinals γ_i such that $\gamma_0 := \gamma(\alpha)$ and $(\dagger) \gamma_{i+1} > \gamma_0 \oplus \gamma_i$.

Let n be the number of projected $(r, l + 2d)$ -Hintikka formulas and set $N := n^2 + 1$. Suppose, for a contradiction, that $\gamma_N \leq \beta$. Then each ordinal $\omega^{\gamma_i} (0 \leq i \leq N)$ has a code, say $w_i \in D$. For all $t > \max_{0 \leq i \leq N} \#w_i$ there exist two indices $c < d \leq N$ such that $\lambda_{w_c,t} = \lambda_{w_d,t}$ and $\rho_{w_c,t} = \rho_{w_d,t}$ (by choice of N). Then

$$\begin{aligned} \omega^{\gamma_d} &\leq \beta_{L,w_d,t} \otimes \beta_{R,w_d,t} \quad (\text{by } \star\star) \\ &< \omega^{\gamma_0} \otimes \beta_{R,w_d,t} \quad (\text{by } \star\star\star) \\ &= \omega^{\gamma_0} \otimes \beta_{R,w_c,t} \quad (\text{by Lemma B}) \\ &\leq \omega^{\gamma_0} \otimes \omega^{\gamma_c} \quad (\text{by } \star) \end{aligned}$$

which equals $\omega^{\gamma_0 \oplus \gamma_c}$, contradicting (\dagger) . Thus $\beta < \gamma_N$ and we have bound the arbitrarily chosen β in terms of r, l, d and $G_1(\alpha')$ for $\alpha' < \alpha$. This achieves the **aim**. We conclude that the $\delta \leq \gamma_N$. Of course as r, l and d vary there is no bound on N . Thus define $G_1(\alpha) := \sup_i \gamma_i$ so that $\delta < G_1(\alpha)$.

To be concrete, take $\gamma_{i+1} := (\gamma_0 \oplus \gamma_i) + 1$, so $\sup_i \gamma_i = \gamma_0 \times \omega$. There are four cases: 1) if $1 \leq \alpha < \omega$ then $\gamma(\alpha) = G_1(\alpha - 1) = \omega^{\alpha-1}$ and so $G_1(\alpha) = \omega^\alpha$; 2) if $\alpha = \omega$ then $\gamma(\omega) = \sup_{n < \omega} \omega^n = \omega^\omega$ and so $G_1(\omega) = \omega^{\omega+1}$; 3) if $\alpha > \omega$ is a successor $\beta + 1$ then $\gamma(\alpha) = G_1(\beta) = \omega^{\beta+1} = \omega^\alpha$ and so $G_1(\alpha) = \omega^{\alpha+1}$; 4) if $\alpha > \omega$ is a limit ordinal then $\gamma(\alpha) = \sup_{\alpha' < \alpha} \omega^{\alpha'+1} = \omega^\alpha$ and so $G_1(\alpha) = \omega^{\alpha+1}$.

Proof of Lemma A. For $t > \#w$, $y, y' \in L_{w,t}$ and $z, z' \in R_{w,t}$ we claim $\text{PrTp}(w, y, y', z)_{[0,\omega]} = \text{PrTp}(w, y, y', z')_{[0,\omega]}$ and $\text{PrTp}(w, y, z, z')_{[0,\omega]} = \text{PrTp}(w, y', z, z')_{[0,\omega]}$. We prove the first equality (the second is similar). Recall that $+$ is the operation summing types of chains. Then $\text{PrTp}(w, y, y', z)_{[0,\omega]}$ equals

$$\begin{aligned} \text{PrTp}(w, y, y', z)_{[0,t)} + \text{PrTp}(w, y, y', z)_{[t,\omega)} &= \\ \text{PrTp}(w, y, y', \emptyset)_{[0,t)} + \text{PrTp}(\emptyset, \emptyset, \emptyset, z)_{[t,\omega)} &= \\ \text{PrTp}(w, y, y', z')_{[0,t)} + \text{PrTp}(\emptyset, \emptyset, \emptyset, z')_{[t,\omega)} &= \\ \text{PrTp}(w, y, y', z')_{[0,t)} + \text{PrTp}(w, y, y', z')_{[t,\omega)} &= \\ \text{PrTp}(w, y, y', z')_{[0,\omega)}. \end{aligned}$$

To go from the second line to the third line use that $\text{PrTp}(\emptyset, \emptyset, \emptyset, z)_{[t,\omega)}$ is determined by $\text{PrTp}(\emptyset, z)_{[t,\omega)}$ which is $\rho_{w,t}$ by definition of $R_{w,t}$. Since $z' \in R_{w,t}$ then also $\text{PrTp}(\emptyset, \emptyset, \emptyset, z')_{[t,\omega)}$ is determined by $\rho_{w,t}$.

Thus for $t > \#w$, we can replace \exists by \forall in the definitions of $\prec_{L,w,t}$ and $=_{L,w,t}$, and $\prec_{R,w,t}$ and $=_{R,w,t}$. For example, if $y, y' \in L_{w,t}$ and $z \in R_{w,t}$ and $y \cup z =_\Gamma y' \cup z$, then for all $z' \in R_{w,t}$ it holds that $y \cup z' =_\Gamma y' \cup z'$. It is now immediate that both $(L_{w,t}, \prec_{L,w,t}) / =_{L,w,t}$ and $(R_{w,t}, \prec_{R,w,t}) / =_{L,w,t}$ are well-defined well-orders.

Proof of Lemma B. First note $R_{w,t} = R_{w',t}$ since $\text{PrTp}(w, z)_{[t,\omega)} = \text{PrTp}(\emptyset, z)_{[t,\omega)} = \text{PrTp}(w', z)_{[t,\omega)}$. Second by the reasoning in Lemma A and using $\lambda_{w,t} = \lambda_{w',t}$ we see that if $y_1, y_2 \in R_{w,t}$ and $z \in L_{w,t}$ with $y_1 \cup z \preceq y_2 \cup z$ then for all $z' \in L_{w',t}$ it holds that $y_1 \cup z' \preceq y_2 \cup z'$.

Case 2. Say $\mathfrak{T} = \sum_{i \in \text{Ind}} \mathfrak{T}_i$ is a finite-sum of type 1 trees each of rank α . We prove $\omega^\delta < \omega^{G_1(\alpha)}$. By Lemma 2.3 (partition property for orders) we may assume that for every $i \in \text{Ind}$ there is a type τ_i such that if $x \in D$ then $(\mathfrak{T}_i, x \cap T_i)$ has type τ_i . Define $D_i := \{x \cap T_i \mid x \in D\}$ and a binary relation \prec_i on D_i by $x \prec_i y$ if $\exists z \in D \ x \cup (z \setminus T_i) \prec y \cup (z \setminus T_i)$, and similarly a binary relation $=_i$. By the same compositional reasoning as above $(D_i, \prec_i) / =_i$ is well-ordered and $\leq \omega^\delta$, say of type η_i . By a fact similar to Proposition 2.18 ordinal η_i is finite-set interpretable in (Case 1 tree) \mathfrak{T}_i , thus $\eta_i < \omega^{G_1(\alpha)}$. The function sending $(x_1, \dots, x_{|\text{Ind}|}) \mapsto \bigcup x_i$ from $D_1 \times \dots \times D_{|\text{Ind}|} \rightarrow D$ induces a surjective co-ordinate wise non-decreasing function $\eta_1 \times \dots \times \eta_{|\text{Ind}|} \rightarrow \omega^\delta$. Thus $\omega^\delta \leq \eta_1 \otimes \dots \otimes \eta_{|\text{Ind}|}$. But since each $\eta_i < \omega^{G_1(\alpha)}$ and since $G_1(\alpha)$ is a power of ω , we see that $\omega^\delta < \omega^{G_1(\alpha)}$. Thus $\text{rank}(\omega^\delta) = \delta < G_1(\alpha)$.

Finally, the same proof goes through for d -dim interpretations (replace variables by tuples of variables and make minor changes in notation). And the dimension d has no effect on G_d ; that is $G_d = G_1$. Thus define $G := G_1$ to complete the proof. ■

The proof just presented can be adapted to scattered rank α trees of height $\omega+1$, in particular to completions $\widehat{\mathfrak{T}}$. We explain the terminology. A ‘well-founded tree’ is one in which every set of the form $\{y \mid y \preceq x\}$ is a (not necessarily finite) well-founded set. The *height* of a well-founded tree is the supremum of the order types of these sets. Thus the trees as defined in Section II-B have height $\leq \omega$. Writing $[\mathfrak{T}]$ for the infinite branches of \mathfrak{T} define the *completion* of a tree \mathfrak{T} , written $\widehat{\mathfrak{T}}$, as the partial order whose domain is $T \cup [\mathfrak{T}]$ and for which u is below v if either $u, v \in T$ and $u \prec \mathfrak{T} v$, or $u \in T, v \in [\mathfrak{T}]$ and $u \in v$ (that is, u is a node on infinite branch v). If \mathfrak{T} has height $\leq \omega$ then $[\mathfrak{T}]$ has height $\leq \omega+1$. To define scattered trees of height $\omega+1$ we replace ω -sums by $\omega+1$ -sums $\sum_{i < \omega+1} \mathfrak{T}_i$ where \mathfrak{T}_ω is a tree with exactly one element.

Corollary 4.3: Let G be the function from Theorem 4.1. No ordinal of rank $\geq G(\alpha)$ is finite-set interpretable in the completion of any labelled tree of rank $\leq \alpha$.

Proof Sketch: The composition theorem holds for well-founded trees, so we can run the proof of Theorem 4.1 with the following modifications: at the start of Case 1, partition the domain depending on whether the set hits T_ω or not. It is sufficient to deal with each of these domains. The latter case is as before. For the former case replace $[t, \omega)$ by $[t, \omega]$, and define $\#s$ as the smallest integer (exclude ω). In Lemma A for instance $\text{PrTp}(w, y, y', z)_{[\omega, \omega]} = \text{PrTp}(w, \emptyset, \emptyset, z)_{[\omega, \omega]}$, which is, now, also independent of the set w . This yields the same function G . ■

V. REPLACING SET-INTERPRETATIONS BY SIMPLER INTERPRETATIONS

If \mathfrak{A} is finite-set interpretable in \mathfrak{T} then \mathfrak{A} is necessarily countable. A general problem, that we do not solve, states:

Problem 5.1: If \mathfrak{A} is countable and set-interpretable in (not necessarily scattered) tree \mathfrak{T} , is \mathfrak{A} finite-set interpretable in \mathfrak{T} ?

Here is our contribution.

Theorem 5.2: For every injective set interpretation Γ there exists injective set interpretation Γ_f such that (for labelled tree \mathfrak{T} that is not necessarily scattered) if $\Gamma \mathfrak{T}$ is countable then

- 1) $\Gamma \mathfrak{T}$ is set interpretable in \mathfrak{T} via Γ_f , and
- 2) every set in every tuple in the domain of $\Gamma_f \mathfrak{T}$ is either a finite subset of T or a finite union of infinite branches of \mathfrak{T} .

Corollary 5.3: Let G be the function from Theorem 4.1. No ordinal of rank $\geq G(\alpha)$ is injectively set-interpretable in any labelled tree of rank $\leq \alpha$.

Proof: Since a finite subset of $\widehat{\mathfrak{T}}$ is, modulo interpretation, a union of finite sets and finitely many infinite branches, Theorem 5.2 states that if \mathfrak{A} is countable and injectively set interpretable in \mathfrak{T} then \mathfrak{A} is finite-set interpretable in the completion $\widehat{\mathfrak{T}}$. Apply Corollary 4.3. ■

Corollary 5.4: Neither $\mathbb{P}(\mathbb{Q})$ nor $\mathbb{P}(\mathfrak{T}_2)$ is injectively set-interpretable in any scattered tree.

Conjecture 5.5: Neither $\mathbb{P}(\mathbb{Q})$ nor $\mathbb{P}(\mathfrak{T}_2)$ is set-interpretable in any scattered tree.

A. Proof Plan

Given an MSO-formula φ the aim is to define an MSO-formula CODE such that for every tree \mathfrak{T} for which $\varphi\mathfrak{T}$ is countable:

- CODE is an injective function with domain $\varphi\mathfrak{T}$,
- the range of CODE consists of tuples whose sets are either finite subsets of T or finite unions of finitely many infinite branches of \mathfrak{T} .

If φ is the domain formula of an injective set-interpretation Γ then define finitary interpretation Γ_f as follows: its domain formula expresses that \overline{X} is in the range of CODE, its i th relation formula, say of arity n , expresses that there exist \overline{Y}_j s such that $\text{CODE}(\overline{Y}_j, \overline{X}_j)$ and $\phi_{\Gamma}^i(\overline{Y}_1, \dots, \overline{Y}_n)$ (where ϕ_{Γ}^i is the i th relation formula in the interpretation Γ). Injectivity ensures that CODE is an isomorphism between $\Gamma\mathfrak{T}$ and $\Gamma_f\mathfrak{T}$.

In section V-B we discuss structural properties of $\varphi\mathfrak{T}$. In section V-C we provide a first coding that when applied to finitely-branching tree \mathfrak{T} codes \overline{V} (for $\mathfrak{T} \models \varphi(\overline{V})$) by a subtree with finitely many (finite and infinite) branches as well as a labelling of this subtree. In section V-D we sketch how to replace the labelling of the finitely many infinite branches by a tuple of finite sets. If the first coding is applied to a countably-branching tree \mathfrak{T} we still obtain a subtree with finitely many infinite branches, but now it may also contain infinitely many finite branches. In the full version of the paper we show how to replace the labelled subtree consisting of the finite branches with a tuple of finite sets.

B. Structural Properties

The definitions and ideas of this section are from [2].

Definition 5.6 (U-trees and D-trees): Let \mathfrak{T} be an l -tree, \overline{V} a k -tuple and r an integer. If there exists $\overline{W} \neq \overline{V}$ (tuples of subsets of T) with $\text{Tp}^r(\mathfrak{T}, \overline{W}) = \text{Tp}^r(\mathfrak{T}, \overline{V})$ then call $(\mathfrak{T}, \overline{V})$ a *D-tree wrt. r, k* . Otherwise call $(\mathfrak{T}, \overline{V})$ a *U-tree wrt. r, k* .

Definition 5.7 (trunk): Define $\text{trunk}^r(\mathfrak{T}, \overline{V})$ as the set of nodes $u \in T$ such that the subtree of $(\mathfrak{T}, \overline{V})$ rooted at u is a *D-tree wrt. r, k* .

Lemma 5.8: The set $\text{trunk}^r(\mathfrak{T}, \overline{V})$ is MSO-definable in $(\mathfrak{T}, \overline{V})$ and downward closed.

We can decompose a tree along a downward closed set:

Definition 5.9 (tree decomposition): Let \mathfrak{T} be an l -tree and $I \subset T$ a downward closed set. For $i \in I$ define T_i as those $t \in T$ such that $i \preceq t$ and there is no $i' \in I$ with $i \prec i' \preceq t$. As usual write \mathfrak{T}_i for the substructure of \mathfrak{T} restricted to T_i . We call the family $\{\mathfrak{T}_i\}_{i \in I}$ the *I-decomposition of \mathfrak{T}* .

If $\{\mathfrak{T}_i\}_{i \in I}$ is the *I-decomposition of \mathfrak{T}* then \mathfrak{T} is isomorphic to $\sum_{i \in I} \mathfrak{T}_i$ and the H_l^r -partition of I induced by $\{\mathfrak{T}_i\}_{i \in I}$ is definable in \mathfrak{T} expanded by I .

Lemma 5.10 (interpretability of H_l^r -expansion): For every r, l there is a 1-dim injective point interpretation Γ such that for every tree \mathfrak{T} and downward closed $I \subset T$ — writing $\{\mathfrak{T}_i\}_{i \in I}$

for the I -decomposition of \mathfrak{T} — $\Gamma(\mathfrak{T}, I)$ is isomorphic to the H_l^r -expansion of (I, \prec) induced by $\{\mathfrak{T}_i\}_{i \in I}$.

Proposition 5.11 (trunk is finitary): Let φ be a formula of quantifier-rank r and \mathfrak{T} a labelled tree. If $\varphi\mathfrak{T}$ is countable then for every \overline{V} satisfying φ in \mathfrak{T} — writing $\{\mathfrak{T}_i\}$ for the $\text{trunk}^r(\mathfrak{T}, \overline{V})$ -decomposition of $(\mathfrak{T}, \overline{V})$ —

- 1) All but finitely many \mathfrak{T}_i s are *U-trees*.
- 2) The set $\text{trunk}^r(\mathfrak{T}, \overline{V})$ is a union of a finite set and a finite set of infinite branches.

C. First Coding

Suppose \mathfrak{T} is an l -tree, $\varphi\mathfrak{T}$ is countable and r is the quantifier-rank of $\varphi(X_1, \dots, X_m)$. For \overline{V} such that $\mathfrak{T} \models \varphi(\overline{V})$ let $\{\mathfrak{T}_i\}$ be the trunk := $\text{trunk}^r(\mathfrak{T}, \overline{V})$ -decomposition of tree $(\mathfrak{T}, \overline{V})$. Write E for finite set of $i \in \text{trunk}$ such that \mathfrak{T}_i is a *D-tree*. Write BUDS for the set of children of the root of \mathfrak{T}_i for $i \in E$. We can code m -tuple \overline{V} by the following data:

- 1) a pair (F, B) where F, B partition trunk, F is a finite set, and B is a finite set of infinite branches,
- 2) the H_{l+m}^r -partition of F induced by $\{\mathfrak{T}_i\}_{i \in F}$,
- 3) the H_{l+m}^r -partition of B induced by $\{\mathfrak{T}_i\}_{i \in B}$,
- 4) the H_{l+m}^r -partition of BUDS induced by $\{(\mathfrak{T}_{\geq s}, \overline{V})\}_{s \in \text{BUDS}}$.

This coding is injective: we argue that the coding of \overline{V} uniquely determines \overline{V} . Consider $j \in \text{trunk}$. If \mathfrak{T}_j is a *U-tree* then $\overline{V} \cap T_j$ is determined by the data in 2) and 3); if \mathfrak{T}_j is a *D-tree* then consider $i \in \mathfrak{T}_{\geq s}$ for some child $s \in \text{BUDS}$ of the root of \mathfrak{T}_j . Then $\overline{V} \cap T_{\geq s}$ is determined by the data in 4) since it is a *U-tree*. Moreover $\overline{V} \cap \{j\}$ is determined by 2) and 3). The coding is MSO-definable: indeed, F, B are definable from trunk which is definable by Lemma 5.8, partitions are definable by Lemma 5.10, and E and BUDS are definable since the set of *D-trees* (wrt. r, k) are definable.

The predicates in 1) and 2) are finitary: F is a finite set, its partition is a tuple of finite sets, and B is a finite set of infinite branches. Two tasks remain.

Task 1. The predicates in 3) label the subtree on domain B . In Section V-D we sketch how to code this labelling by a tuple of finite sets.

Task 2. If \mathfrak{T} is finitely-branching then each predicate in 4) is a finite set. In the full version we show how, if \mathfrak{T} is countably-branching, to code the possibly infinite set BUDS and its H_{l+m}^r -partition by a tuple of finite sets.

D. Dealing with Infinite Labelled Branches (Task 1)

Proposition 5.12: [2] For Δ_l -formula $\varphi(X_1, \dots, X_m)$ there is ψ such that for every l -tree \mathfrak{T} and every branch I_b of \mathfrak{T} — writing $\mathcal{C} = (I_b, \prec, \overline{Q})$ for the H_l^{l+m} -expansion of I_b induced by the I_b -decomposition of \mathfrak{T} — the following holds for all \overline{W} : $\mathcal{C} \models \psi(\overline{W})$ if and only if there exists \overline{V} such that

- 1) $\mathfrak{T} \models \varphi(\overline{V})$, and
- 2) \overline{W} is the H_{l+m}^r -partition of I_b induced by the I_b -decomposition of $(\mathfrak{T}, \overline{V})$.

In particular if $\varphi\mathfrak{T}$ is countable then $\psi\mathcal{C}$ is countable.

Definition 5.13: For sets $X, Y \subset T$, write $X =_{\text{end}} Y$ to mean that the symmetric difference of X and Y is finite (and

say that X and Y have the same *end*). This notion extends to k -tuples: write $\overline{X} =_{\text{end}} \overline{Y}$ if $X_i =_{\text{end}} Y_i$ for all $i \leq k$.

Proposition 5.14 (definable ends in ω): For every Δ_s -formula $\psi(X_1, \dots, X_n)$ there exist a constant $M := M(s, n)$ and formulas $\Psi_1(\overline{X}), \dots, \Psi_M(\overline{X})$ such that for every s -chain \mathcal{C} over ω there exist M -many tuples $\overline{W}_1, \dots, \overline{W}_M$ such that if $\varphi\mathcal{C}$ is countable then

- 1) $\mathcal{C} \models \psi(\overline{V})$ implies there is $i \leq M$ with $\overline{W}_i =_{\text{end}} \overline{V}$.
- 2) the only tuple satisfied by Ψ_i in \mathcal{C} is \overline{W}_i .

The first item appears in [16]. The second uses the selection property for ω -chains: a formula $\alpha(\overline{X})$ is a selector for formula $\beta(\overline{X})$ over a class of structures \mathcal{C} if the following conditions hold in \mathcal{C} : 1) there is at most one \overline{X} with $\alpha(\overline{X})$; 2) for all \overline{X} if $\alpha(\overline{X})$ then $\beta(\overline{X})$; 3) if there exists \overline{Y} with $\beta(\overline{Y})$ then there exists \overline{X} with $\alpha(\overline{X})$. Every MSO-formula β has a selector α , also an MSO-formula, over the class of all expansions of ω by unary predicates, see [17], [22]. Since a branch of \mathfrak{T} is isomorphic to ω , from Propositions 5.12 and 5.14, and Lemma 5.10 we get:

Proposition 5.15 (definable ends along a branch): For every Δ_l -formula $\varphi(X_1, \dots, X_m)$ of quantifier rank r there exists a constant M and MSO-formulas Φ_1, \dots, Φ_M such that for every l -tree \mathfrak{T} with $\varphi\mathfrak{T}$ countable, if $I_b \subset T$ is an infinite branch of \mathfrak{T} and $\{\mathfrak{T}_i\}_{i \in I_b}$ is the I_b -decomposition of \mathfrak{T} then there exist M -many tuples $\overline{W}_1, \dots, \overline{W}_M$ over I_b such that

- 1) $\mathfrak{T} \models \varphi(\overline{V})$ implies some \overline{W}_j has the same end as the H_{l+m}^r -partition of I_b induced by $\{(\mathfrak{T}_i, \overline{V})\}_{i \in I_b}$.
- 2) \overline{W}_i is the unique tuple satisfied by Φ_i in (\mathfrak{T}, I_b) .

We sketch how to finish Task 1. Recall we have to encode, by a tuple of finite sets, the H_{l+m}^r -partition of B induced by $\{\mathfrak{T}_i\}_{i \in B}$ (where $\{\mathfrak{T}_i\}$ is the trunk^r($\mathfrak{T}, \overline{V}$)-decomposition of $(\mathfrak{T}, \overline{V})$). One set stores, for each of the finitely many branches I in B , an index $n \leq M$ such that the H_{l+m}^r -partition of I induced by $\{\mathfrak{T}_i\}_{i \in I}$ has the same end as tuple defined by Φ_n . The same set stores from which point of I onwards the tuples agree. In fact the index for I can be coded as a label of a definable node y of I that is on no other branch of B (see formula ϵ below). Also mark the \prec -least node z of I above y from which point on the tuples agree. Finally we need to store the restriction of the partition to all nodes below z . This data can be stored in a tuple of finite sets, and determines the H_{l+m}^r -partition of B . We now argue that it is MSO-definable.

Formally, apply Proposition 5.15 to the domain formula of Γ . This gives M and Φ_1, \dots, Φ_M . For $n \leq M$ define formula $\mu_n(\overline{V}, I)$ stating that I is an infinite branch of trunk^r($\mathfrak{T}, \overline{V}$) and n is the least integer with the property that the unique tuple \overline{W}_n over I satisfying Φ_n has the same end as the H_{l+m}^r -partition of I induced by $\{\mathfrak{T}_i\}_{i \in I}$; if furthermore \overline{W}_n and the mentioned H_{l+m}^r -partition of I agree on $\{i \in I \mid z \preceq i\}$ then write $\nu_n(\overline{V}, I, z)$. Define an auxiliary formula $\epsilon(X, x)$ stating that X is an infinite branch and x is the \prec -minimal element such that x is on X and no two elements of B above x are \prec -incomparable.

Finally, code the H_{l+m}^r -partition of B induced by $\{\mathfrak{T}_i\}_{i \in B}$ by $|H_{l+m}^r|$ -tuple of finite sets \overline{H} and M -tuple of finite sets \overline{G} :

- 1) For every $n \leq M$: $z \in G_n$ if and only there exists I, y such that $\mu_n(\overline{V}, I)$ and $\epsilon(I, y)$ and z is the \prec -minimal element such that $y \preceq z$ and $\nu_n(\overline{V}, I, z)$;
- 2) \overline{H} is the restriction of the H_{l+m}^r -partition of B induced by $\{\mathfrak{T}_i\}_{i \in B}$ to the finite set $\bigvee_{n \leq M} \{u \mid \exists z \in G_n u \preceq z\}$.

ACKNOWLEDGMENTS

Vince Bárány, Lukasz Kaiser, Phillip Schlict, funding from the European Science Foundation for the activity entitled ‘Games for Design and Verification’, FWF NFN Nr. S11407-N23 (RiSE), ERC 279307: Graph Games, PROSEED WWTF Nr. ICT 10-050, ARISE FWF Nr. S11403-N23.

REFERENCES

- [1] F. Abu Zaid, E. Grädel, and L. Kaiser, “The field of reals is not omega-automatic,” in *STACS*, 2012.
- [2] V. Bárány, L. Kaiser, and A. Rabinovich, “Expressing cardinality quantifiers in monadic second-order logic over trees,” *Fundamenta Informaticae*, vol. 100, pp. 1–18, 2010.
- [3] V. Bárány, L. Kaiser, and S. Rubin, “Cardinality and counting quantifiers on omega-automatic structures,” in *STACS*, 2008.
- [4] A. Blumensath and E. Grädel, “Automatic structures,” in *15th Symposium on Logic in Computer Science (LICS)*, 2000, pp. 51–62.
- [5] P. W. Carruth, “Arithmetic of ordinals with applications to the theory of ordered abelian groups,” *Bull. AMS*, vol. 48, pp. 262–271, 1942.
- [6] D. Caucal, “On infinite terms having a decidable monadic theory,” in *MFCS*, ser. Lecture Notes in Comput. Sci., vol. 2420, 2002, pp. 165–176.
- [7] T. Colcombet and C. Löding, “Transforming structures by set interpretations,” *Logical Methods in Computer Science*, vol. 3, no. 2, 2007.
- [8] B. Courcelle, “The monadic second-order logic of graphs ix: Machines and their behaviours,” *Theoretical Computer Science*, vol. 151, no. 1, pp. 125–162, 1995.
- [9] C. Delhommé, “Automaticité des ordinaux et des graphes homogènes,” *Comptes Rendus Mathématique*, vol. 339, no. 1, pp. 5–10, 2004.
- [10] C. Elgot and M. Rabin, “Decidability of extensions of theory of successor,” *J. Symb. Log.*, vol. 31, no. 2, pp. 169–181, 1966.
- [11] Y. Gurevich, “Monadic second-order theories,” in *Model-Theoretic Logics*. Springer, 1985, pp. 479–506.
- [12] Y. Gurevich and S. Shelah, “Rabin’s uniformization problem,” *J. Symb. Log.*, vol. 48, pp. 1105–1119, 1983.
- [13] ———, “On the strength of the interpretation method,” *J. Symb. Log.*, vol. 54, no. 2, pp. 305–323, 1989.
- [14] W. A. Hodges, *Model Theory*. Cambridge University Press, 1993.
- [15] B. Khousainov and A. Nerode, “Automatic presentations of structures,” *Lecture Notes in Computer Science*, vol. 960, pp. 367–392, 1995.
- [16] D. Kuske and M. Lohrey, “First-order and counting theories of omega-automatic structures,” *J. Symb. Log.*, vol. 73, no. 1, pp. 129–150, 2008.
- [17] S. Lifsches and S. Shelah, “Uniformization and skolem functions in the class of trees,” *J. Symb. Log.*, vol. 63, pp. 103–127, 1998.
- [18] ———, “Random graphs in the monadic theory of order,” *Archive for Math Logic*, pp. 273–312, 1999.
- [19] A. Nies, “Describing groups,” *Bull. Symb. Log.*, vol. 13, no. 3, pp. 305–339, 2007.
- [20] M. Rabin, “Decidability of second-order theories and automata on infinite trees,” *AMS Transactions*, vol. 141, pp. 1–35, 1969.
- [21] A. Rabinovich, “The full binary tree cannot be interpreted in a chain,” *J. Symb. Log.*, vol. 75, pp. 1489–1498, 2010.
- [22] ———, “On decidability of monadic logic of order over the naturals extended by monadic predicates,” *Inf. Comput.*, vol. 205, no. 6, pp. 870–889, 2007.
- [23] J. Rosenstein, *Linear Orderings*. Academic Press, 1982.
- [24] S. Rubin, “Automata presenting structures: a survey of the finite string case,” *Bull. Symb. Log.*, vol. 14, no. 2, pp. 169–209, 2008.
- [25] S. Shelah, “The monadic theory of order,” *Annals of Mathematics*, pp. 349–419, 1975.
- [26] W. Thomas, “Constructing infinite graphs with a decidable MSO-theory,” in *MFCS 2003*, ser. LNCS, vol. 2747, 2003, pp. 113–124.
- [27] T. Tsankov, “The additive group of the rationals does not have an automatic presentation,” *JSL*, vol. 76, no. 4, pp. 1341–1351, 2011.

How to Travel between Languages^{*}

Krishnendu Chatterjee¹ and Siddhesh Chaubal² and Sasha Rubin^{1,3}

¹ IST Austria, Am Campus 1, 3400 Klosterneuburg, Austria
`Krishnendu.Chatterjee@ist.ac.at, sasha.rubin@ist.ac.at`

² IIT Bombay, Powai, Mumbai 400076, India `spchaubal@gmail.com`

³ TU Wien, Institut für Informationssysteme 184/4, Favoritenstraße 911, 1040, Vienna, Austria

Abstract. We consider how to edit strings from a source language so that the edited strings belong to a target language, where the languages are given as deterministic finite automata. Non-streaming (or offline) transducers perform edits given the whole source string. We show that the class of deterministic one-pass transducers with registers along with increment and min operation suffices for computing optimal edit distance, whereas the same class of transducers without the min operation is not sufficient. Streaming (or online) transducers perform edits as the letters of the source string are received. We present a polynomial time algorithm for the *partial-repair problem* that given a bound α asks for the construction of a deterministic streaming transducer (if one exists) that ensures that the ‘maximum fraction’ η of the strings of the source language are edited, within cost α , to the target language.

Keywords: Edit-distance, Markov decision process, register automaton

1 Introduction

One of the classic problems in language theory concerns optimally editing an input string so that it belongs to a given target regular language [3].

Definition 1 (Edit-distance). An edit operation applied to a string u either deletes a single character, inserts a single character, or changes a single character. The edit-distance between $u, v \in \Sigma^*$, denoted $\text{ED}(u, v)$, is defined as the length of a shortest sequence of edit operations that applied to u yields v . For language T , we define $\text{ED}(u, T) := \inf_{v \in T} \text{ED}(u, v)$.

In [2] the problem was generalized: given source language R and target language T , how to edit strings from R so that the edited strings belong to T .

* The research was supported by Austrian Science Fund (FWF) Grant No P 23499-N23, FWF NFN Grant No S11407-N23 (RiSE), ERC Start grant (279307: Graph Games), and Microsoft faculty fellows award. Thanks to Gabriele Puppis for suggesting the problem of identifying a deterministic transducer to compute the optimal cost, and to Martin Chmelik for his comments on the introduction.

There are two broad categories of transducers \mathfrak{T} that edit strings depending on how they process input. The *non-streaming (or offline)* transducers reads the complete source string $u \in R$ and then output the repaired string $\mathfrak{T}(u) \in T$. The *streaming (or online)* transducers perform the edits as the letters of input u are received. For a class of transducers, the main interest is to compare $u \mapsto \text{ED}(u, \mathfrak{T}(u))$ — ie. the cost of editing using \mathfrak{T} from the class — to $u \mapsto \text{ED}(u, T)$, the cost of optimal editing.

Previous results for non-streaming transducers: Wagner [3] gives a polynomial time algorithm for computing the optimal edit-distance, namely $u \mapsto \text{ED}(u, T)$, given a DFA for T . In [2] it was shown (Section III.A) that certain (non-deterministic) distance automata can compute this optimal cost.

Our contributions for non-streaming transducers: We consider the problem of finding a natural class of *deterministic* transducer for computing the optimal edit-distance $u \mapsto \text{ED}(u, T)$. We observe that Wagner’s algorithm can be reformulated using cost-register automata of [1]. Specifically, one can use the deterministic one-pass transducers with registers that allow parallel updates using the increment and arbitrary-arity minimum operations. We prove that natural restrictions of this model, notably by disallowing use of the minimum operator, do not suffice to compute the optimal edit-distance (Theorem 5). This uses some pumping-like arguments.

Previous results for streaming transducers: In [2] it was also shown that whether there is a streaming transducer (Definition 7) with finite streaming-cost (Definition 8) that repairs strings from R to strings in T is a PTIME-complete problem (the languages R, T are given as DFAs). Moreover, if the cost is finite, then a streaming transducer can be extracted from their proof [2][Theorem 3].

Our contributions for streaming transducers: We consider the problem of repairing strings from source to target language in the case that the streaming-cost is infinite or very large. In this case we fix an edit-distance bound α and ask what is the ‘largest fraction’ of strings $\eta \in [0, 1]$ that can be repaired within cost α by a streaming transducer. This is called the *partial-repair problem*. We show with an example (Example 12) that although the streaming-cost is infinite, a large fraction (formalised in Definition 11) of the strings from the source language may be edited with small edit distance to belong to the target language. Our main contribution (Theorem 14) is a polynomial time algorithm solving the partial-repair problem, and, if one exists, a construction of a corresponding deterministic streaming transducer. We do this by building and solving a Markov decision process whose value is the largest such η .

2 Non-streaming Transducers

Following Wagner [3], there is a dynamic programming algorithm that given a string $u \in \Sigma^*$ and a DFA for target language $T \subset \Sigma^*$ can compute, in PTIME, the integer $\text{COST}(u, T)$ (and a string $t \in T$ with the property that $\text{COST}(u, T) = \text{ED}(u, t)$). In [2][Section III] it is mentioned that this gives a transducer of fairly low complexity. We formalise this intuition and observe that

there is a *deterministic* transducer model that implements Wagner's approach and computes $u \mapsto \text{ED}(u, T)$. In fact, the transducers are certain cost register automata (introduced in [1]), which we call *repair-transducers*. A repair-transducer is a DFA with a fixed number of register names K . Write int_k for the value, a natural number, of register $k \in K$. Initially each register contains 0. At each step the DFA reads the next letter from the input string, updates its local state, and makes a parallel update to the registers. The allowed updates may mention the register names, the constant 0, addition by a constant, and the minimum of any number of terms.¹ Once the input string has been read the machine outputs the contents of some of its registers. Thus a repair-transducer realises a function $\Sigma^* \rightarrow \mathbb{N}^k$. Formally, an update is a term generated by the grammar:

```
inc-min ::= 0 | int_k (for k in K) | inc-min + c (for c in N)
inc-min ::= min(inc-min, inc-min, ..., inc-min)
```

If $\nu : K \rightarrow \mathbb{N}$ and τ is an inc-min term, write $[[\tau]]_\nu$ for the evaluation of term τ under assignment ν .²

Definition 2. A repair-transducer \mathfrak{T} is a DFA (Σ, Q, q_0, δ) without final states augmented by a finite set K of register names, a register update function $\mu : Q \times \Sigma \times K \rightarrow \text{IMT}$, and a final register function $f : Q \rightarrow K$, where IMT is the set of inc-min terms. A configuration is an element of $Q \times \mathbb{N}^K$. The initial configuration is (q_0, ν_0) where $\nu_0(k) = 0$ for all $k \in K$. The run on input $u = u_1 \cdots u_n \in \Sigma^*$ is the sequence of configurations $(q_0, \nu_0) \cdots (q_n, \nu_n)$ such that $q_{i+1} = \delta(q_i, u_i)$ (for $0 \leq i < n$) and for each $k \in K$, $\nu_{i+1}(k) := [[\mu(q_i, u_i, k)]]_{\nu_i}$. The transducer outputs $\mathfrak{T}(u) := \nu_n(f(q_n))$.

Proposition 3. For regular language T there is a repair-transducer \mathfrak{T} computing $u \mapsto \text{COST}(u, T)$. Moreover, given a DFA for T one can build \mathfrak{T} in PTIME.

For the proof simply note that the dynamic-programming identities in Wagner's algorithm only use the $+c$ and \min operations. Clearly if we disallow use of $+c$ operation then the transducer can't accumulate values and so can't compute $u \mapsto \text{ED}(u, T)$. What if we disallow the \min operation?

Proposition 4. There is a language T such that every repair-transducer for T requires the use of \min in its update rules.

Proof. Let $T = 0^* + 1^*$. Suppose there were a repair transducer \mathfrak{T} for T with state set Q , register set K , but no use of the \min operation. Let $\delta : Q \times \Sigma^* \rightarrow Q$ be the transition function (extended to strings) and $M : Q \times \Sigma^* \times \mathbb{N}^K \rightarrow \mathbb{N}^K$ the evaluated update function extended to strings. That is: the run of \mathfrak{T} starting from (q, ν) on input $u \in \{0, 1\}^*$ ends in $(\delta(q, u), M(q, u, \nu))$. Note that $\mathfrak{T}(w)$ is the minimum of the number of 0s in w and the number of 1s in w .

¹ This is similar to the inc-min grammar of [1]. However there they only allow a binary \min operation.

² Namely, $[[0]]_\nu := 0$, $[[\text{int}_k]]_\nu := \nu(k)$, $[[\tau + c]] := [[\tau]] + c$, and $[[\min(\tau_1, \dots, \tau_n)]] := \min\{[[\tau_1]], \dots, [[\tau_n]]\}$.

Fact 1. For every (q, u) there are functions $F_{q,u} : K \rightarrow K$ and $\#F_{q,u} : K \rightarrow \mathbb{N}$ such that for all ν, k ,

$$M(q, u, \nu)(k) = \nu(F_{q,u}(k)) + \#F_{q,u}(k), \quad (1)$$

in words: the k th component of $M(q, u, \nu)$ is equal to the $F_{q,u}(k)$ th component of ν plus integer $\#F_{q,u}(k)$. This can be proved by induction on $|u|$ and uses the fact that \mathfrak{T} only has $+c$ updates.

Fact 2. For every $F_{q,u} : K \rightarrow K$ there exists $N, P \in \mathbb{N}$ such that for all $y \in \mathbb{N}$, $(F_{q,u})^N = (F_{q,u})^{N+Py}$. This follows from the fact that the set of functions $K \rightarrow K$ is a finite set closed under composition so apply the pigeonhole principle.

Fact 3. Fix u, q such that $\delta(q, u) = q$, and N, P from Fact 1 applied to $F_{q,u}$. For every $k \in K$ there exists $j \in K$ and $\alpha, \beta \in \mathbb{N}$ such that for all $\nu \in \mathbb{N}^K$ and $y \in \mathbb{N}$, $M(q, u^{N+Py}, \nu)(k) = \beta + y\alpha + \nu(j)$. For the proof use: $\beta := \#F_{q,u^N}(k)$, $j := F_{q,u}^N(k)$, and $\alpha := \#F_{q,u^P}(j)$.

We now apply some pumping arguments. Note that in the lemma if $\alpha = 0$ then pumping doesn't increase the value. In this case call the triple (u, q, k) *flat*. On the other hand if $\alpha > 0$ then pumping increases the value. In this case call the triple (u, q, k) *increasing*. We exploit this dichotomy.

By the pigeonhole principle there exists $q_0 \in Q$ and $c, d \in \mathbb{N}$ such that $\delta(\iota, 0^a) = q_0$, $\delta(q_0, 0^b) = q_0$. Thus $\delta(\iota, 0^{a+bx}) = q_0$ for all $x \in \mathbb{N}$. Similarly, there exists $q_1 \in Q$, $c, d \in \mathbb{N}$ such that $\delta(q_0, 1^c) = q_1$, $\delta(q_1, 1^d) = q_1$. Thus $\delta(q_1, 0^{c+dy}) = q_1$ for all $y \in \mathbb{N}$. Write $w_{x,y} := 0^{a+bx}1^{c+dy}$. Note that $\mathfrak{T}(w_{x,y}) = \min\{a+bx, c+dy\}$. Let $k := f(q_1)$ be the register whose value is output when given strings of the form $w_{x,y}$.

Suppose x is much bigger than y (technically: $a+bx > c+d(y+1)$). Then after reading input $w_{x,y}$ register k has value $c+dy$. And after reading input $w_{x,y+1}$ register k has value $c+d(y+1)$. This implies that the triple $(1^d, q_1, k)$ is increasing. On the other hand, suppose x is smaller than y . Then after reading input $w_{x,y}$ register k has value $a+bx$. And after reading input $w_{x,y+1}$ register k has value $a+bx$. This implies that the triple $(1^d, q_1, k)$ is flat. But a triple can not be both increasing and flat. \square

We summarise the results of this section:

Theorem 5. *For every regular language $T \subset \Sigma^*$ there is a repair-transducer \mathfrak{T} computing $\text{COST} : \Sigma^* \rightarrow \mathbb{N}, u \mapsto \text{COST}(u, T)$. Moreover, the models of transducer which disallow the $+c$ operators or the min operator cannot, in general, compute this cost.*

3 Streaming Transducers

Here is the generalisation of edit-distance to a source and target language:

Definition 6 (Repair-cost). [2] Given two languages $R, T \subset \Sigma^*$ define the repair-cost from R to T as $\text{COST}(R \rightsquigarrow T) := \sup_{u \in R} \inf_{v \in T} \text{ED}(u, v)$.

Note the asymmetry in the definition of repair cost: it expresses the worst-case cost of repairing all strings in R to strings in T . The cost may be finite $\text{COST}((ab)^* \rightsquigarrow (ba)^*) = 2$ — just delete the first and last letter of the input — or infinite $\text{COST}(a^* \rightsquigarrow (ba)^*) = \infty$ — since $\text{ED}(a^{2n}, (ba)^*) = n$.

Definition 7 (Streaming Transducer). A streaming transducer is a device of the form $\mathfrak{T}\mathbf{r} = (\Sigma, \Sigma_{out}, Q, \delta, q_0, \Omega)$, where

- Σ is a finite input alphabet, and Σ_{out} is a finite output alphabet,
- Q is a finite set of states and $q_0 \in Q$ is an initial state,
- δ is a transition function $Q \times \Sigma \rightarrow \Sigma_{out}^* \times Q$, and
- Ω is a final-output function $Q \rightarrow \Sigma_{out}^*$.

For every string $u = a_1 \dots a_n$ in Σ^* , there is a unique sequence of states q_0, q_1, \dots, q_n and strings v_1, \dots, v_n such that $\delta(q_i, a_{i+1}) = (v_{i+1}, q_{i+1})$ for all $0 \leq i < n$. In this case define the output of $\mathfrak{T}\mathbf{r}$ on u to be the string $\mathfrak{T}\mathbf{r}(u) = v_1 v_2 \dots v_n v_{n+1}$ where $v_{n+1} = \Omega(q_n)$. We write $q_0 \xrightarrow{a_1/v_1} q_1 \xrightarrow{a_2/v_2} \dots \xrightarrow{a_n/v_n} q_n \xrightarrow{v_{n+1}}$.

Define the output of $\mathfrak{T}\mathbf{r}$ on language R to be the set $\mathfrak{T}\mathbf{r}(R) = \{\mathfrak{T}\mathbf{r}(u) \mid u \in R\}$.

Definition 8 (Streaming Cost³). [2] For a streaming transducer $\mathfrak{T}\mathbf{r}$ and an input string $u = a_1 \dots a_n \in \Sigma^*$, if the run of u on $\mathfrak{T}\mathbf{r}$ is $q_0 \xrightarrow{a_1/v_1} q_1 \xrightarrow{a_2/v_2} \dots \xrightarrow{a_n/v_n} q_n \xrightarrow{v_{n+1}}$, then the streaming cost of $\mathfrak{T}\mathbf{r}$ on u is defined as:

$$\text{COST}_{\mathfrak{T}\mathbf{r}}(u) = |v_{n+1}| + \sum_{i=1}^n \text{ED}(a_i, v_i)$$

For language R define $\text{COST}_{\mathfrak{T}\mathbf{r}}(R) := \sup_{u \in R} \text{COST}_{\mathfrak{T}\mathbf{r}}(u)$.

If $\mathfrak{T}\mathbf{r}(R) \subset T$ then say that $\mathfrak{T}\mathbf{r}$ is a streaming transducer from R to T .

Note. $\text{ED}(u, \mathfrak{T}\mathbf{r}(u)) \leq \text{COST}_{\mathfrak{T}\mathbf{r}}(u)$ and so $\text{COST}(R \rightsquigarrow \mathfrak{T}\mathbf{r}(R)) \leq \text{COST}_{\mathfrak{T}\mathbf{r}}(R)$. The streaming-cost is an upper bound on the repair-cost. So if $\text{COST}(R \rightsquigarrow T) = \infty$ then there is no streaming-transducer from R to T with finite streaming-cost.

Example 9. [2] Let $\Sigma = \{a, b, c\}$, $R = (a+b)c^*(a^+ + b^+)$ and $T = ac^*a^+ + bc^*b^+$. Then for every $r \in R$ there is $t \in T$ such that $\text{ED}(r, t) \leq 1$ (correct the first letter if required). However, for every streaming transducer $\mathfrak{T}\mathbf{r}$ from R to T , $\text{COST}_{\mathfrak{T}\mathbf{r}}(R) = \infty$. In other words, the cost of repairing all strings in R to strings in T is finite, but is not realisable by a streaming transducer with finite streaming-cost.

In the last example consider a streaming transducer $\mathfrak{T}\mathbf{r}$ that outputs an 'a' and then copies the rest of the input (ie. it sends $(a+b)c^n w$ to $ac^n w$). It correctly repairs strings of the form $(a+b)c^*a^+$ (incurring cost ≤ 1) and incorrectly strings of the form $(a+b)c^*b^+$; that is, it is a streaming transducer from $(a+b)c^*a^+$

³ In [2] this is called the *aggregate cost* of $\mathfrak{T}\mathbf{r}$ on u .

to T . Then, informally, \mathfrak{T} repairs with cost at most 1 half the strings of R to T ; and formally \mathfrak{T} is a $(\frac{1}{2}, 1)$ -streaming transducer (Definition 11). To formalise this we introduce probability measures over infinite strings.

A *distribution on finite set A* is a function $d : A \rightarrow [0, 1]$ with $\sum_{a \in A} d(a) = 1$. For instance, $d : \sigma \mapsto \frac{1}{|A|}$ is a distribution on A , and $d' : \sigma_1 \cdots \sigma_n \mapsto \prod_i d(\sigma_i)$ is a distribution on A^n . The distributions over A will be denoted $\text{dbn}(A)$. For $u \in A^*$, let $\text{cone}(u) \subset A^\omega$ be the set of infinite strings that have u as a prefix. There is a unique *probability measure* μ_d on the Borel σ -field generated by the cones⁴ with the property that the measure of $\text{cone}(u)$ is $d'(u)$.

Example 10. Continuing with Example 9, let d and d' be as above (thus $d'(u) := 3^{-|u|}$). The probability (wrt. μ_d) that every prefix of an infinite string is in $(a+b)c^*a^+$ conditioned on the infinite string having a prefix in $R = (a+b)c^*(a^++b^+)$ is $\frac{1}{2}$.

Definition 11 ((η, α)-streaming transducer). *Fix regular languages R, T , and a streaming transducer \mathfrak{T} from R to T , and a non-negative integer α . Say that infinite string $u = a_0a_1a_2\dots$*

1. is in need of R -repair if there exists n so that $a_0 \dots a_n$ is in R ;
2. is $\langle R, T \rangle$ -repairable by \mathfrak{T} within α if for all n with $a_0 \dots a_n \in R$ the cost $\text{COST}_{\mathfrak{T}}(a_0 \dots a_n)$ is at most α and $\mathfrak{T}(a_0 \dots a_n) \in T$.

Say that \mathfrak{T} is an (η, α) -streaming transducer (from R to T) if η is the probability that u is $\langle R, T \rangle$ -repairable by \mathfrak{T} within α conditioned on u being in need of R -repair. Here probabilities are taken with respect to μ_d induced by the measure on cones d' itself determined by $d : w \mapsto |\Sigma|^{-|w|}$ where Σ is the alphabet of R . When R and T are fixed we may not mention them.

We give an example where η is close to 1:

Example 12. Let $\Sigma = \{a, b, c, d\}$ and $R_k := (\{a, b\}^k \setminus b^k)c^+ \cup b^kd^+$ and $T = (a+b)^*c^+$. Fix k and note that $\text{COST}(R_k \rightsquigarrow T) = \infty$ since $\text{ED}(b^kd^n, T) = n$ (as T does not accept any string with d in it). However, there exists an (η, α) -streaming transducer with $\alpha = 0$ and $\eta = 1 - \frac{1}{2^k}$ which operates as follows: it copies the first k letters and then outputs a ‘c’ for every remaining input letter. The only strings in R_k which it cannot repair within cost 0 are the ones of the form b^kd^+ .

Partial-repair Problem. The *bounded-repair problem* is, given DFAs for R and T to decide whether or not there exists a streaming transducer \mathfrak{T} from R to T such that $\text{COST}_{\mathfrak{T}}(R)$ is finite. It is proved in [2] that the bounded repair problem is PTIME-complete. Moreover, if it exists, a streaming transducer can be constructed quite easily from their proof.

⁴ The Borel σ -field is defined as the least collection of subsets of A^ω containing the cones and closed under countable union and complementation. Sets in the σ -field are called measurable. All our sets in this paper are measurable.

Question 13. Suppose $\text{COST}(R \rightsquigarrow T) = \infty$, or $\text{COST}_{\mathfrak{T}\mathfrak{r}}(R)$ is ∞ or just very large for every streaming-transducer $\mathfrak{T}\mathfrak{r}$ from R to T . How to transform R to T ?

Our proposal is, given R, T and allowed cost α , to construct a streaming transducer $\mathfrak{T}\mathfrak{r}$ that, roughly, repairs as many strings as possible. Formally this means solving the *partial-repair problem*: compute the largest η for which there exists a (η, α) -streaming transducer from R to T ; and compute the corresponding transducer. The main theorem of this section states that we can do this in PTIME:

Theorem 14 (partial-repair problem). *Given DFAs for R and T , and positive integer α , given in unary, one can compute, in PTIME, the largest $\eta \in [0, 1]$ for which there exists an (η, α) -streaming transducer sending R to T .⁵ Moreover, we can build an (η, α) -streaming transducer from R to T in PTIME.*

The rest of the paper is devoted to a proof of this theorem.

3.1 Tools for Theorem 14

Definition 15 (MC). A Markov chain M is a tuple (Q, Δ, ι) where

- Q is a finite set of states,
- $\Delta : Q \rightarrow \mathbf{dbn}(Q)$ gives the transition probabilities, and
- $\iota \in \mathbf{dbn}(Q)$ is the initial distribution.

The edges E consist of pairs (q, q') such that $\Delta(q)(q') > 0$. A path $q_1 q_2 \dots$ of M is a (finite or infinite) sequence of states such that $\iota(q_1) > 0$ and successive states q_i, q_{i+1} satisfy E .

Write Ω_M for the set of infinite paths in M . Form a topology on Ω_M by taking as basis the sets of the form $\mathbf{cone}(x)$ where $\mathbf{cone}(x)$ consists of all infinite paths in M that start with the finite path x . Define the probability in M of a path $q_1 \dots q_n \in Q^+$ as $\iota(q_1) \times \prod_{1 \leq i < n} \Delta(q_i)(q_{i+1})$. Define μ_M on $\mathbf{cone}(x)$ as the probability in M of path x . Then μ_M can be uniquely extended to the Borel σ -field generated by the open sets. Write \Pr_M for the unique probability measure (over Ω_M) extending μ_M .

Definition 16 (Labelled MC). A Markov chain $M = (Q, \Delta, \iota)$ is Σ -labelled if for each $q \in Q$ the edges out of q (ie. $E(q) := \{(q, q') : E(q, q')\}$) are in bijection with Σ .

Being labelled means that every state q has exactly $|\Sigma|$ edges, and each edge goes to a different state.

Example 17 (Uniform MC U_Σ). Let $U = U_\Sigma$ have states Σ , transition from σ to σ' labelled σ' with probability $\frac{1}{|\Sigma|}$, initial distribution sends σ to $\frac{1}{|\Sigma|}$. Then U is a Σ -labelled MC. The probability of $u \in \Sigma^+$ is equal to $|\Sigma|^{-|u|}$. Thus the measure \Pr_U agrees with μ_d on the cones $\mathbf{cone}(u)$. Hence \Pr_U and μ_d agree on the measurable subsets of Σ^ω .

⁵ That is, η is a rational and the algorithm computes a representation for it in PTIME.

The following lemma is standard:

Lemma 18. *There is a PTIME algorithm that given a MC M and a set of states A computes the probability that a path in M reaches A .*

The following definition annotates a MC by the states of a DFA.

Definition 19 ($M \oslash_{mc} D$). *For Σ -labelled Markov chain M and DFA D over alphabet $\Sigma \times Q_M$ define the Σ -labelled Markov chain $M \oslash_{mc} D$ as follows:*

- The state set is $Q_M \times Q_D$.
- Suppose there is an edge $E_M(m, m')$ labelled σ . Then there is an edge from (m, d) to $(m', \delta_D(d, (\sigma, m')))$ with probability $\Delta(m)(m')$ and label σ .
- the initial distribution sends (m, d) to $\iota_M(m)$ if d is the initial state of D , and to zero otherwise;

As a degenerate case, in case D has alphabet Σ then write $M \oslash_{mc} D$ to mean $M \oslash_{mc} F$ where F has the same state set as D , the same initial state, has alphabet $\Sigma \times Q_M$, and sends, for all m , state q on input (σ, m) to $\delta_D(q, \sigma)$.

Note. It can be checked that the object defined is indeed a Σ -labelled Markov chain. We point out that in an edge (m, d) to (m', d') labelled σ , the state d' depends directly on m' — not m — and σ .

Let M and D be as in the definition. Every path $m_1 m_2 \dots$ of M induces a unique sequence of labels $\sigma_1 \sigma_2 \dots$ such that the edge from (m_i, m_{i+1}) is labelled σ_i which itself induces a unique sequence $d_1 d_2 \dots$ of states of D satisfying $d_{i+1} = \delta_D(d_i, (\sigma_i, m_{i+1}))$ where d_1 is the initial state of D . Let

$$\rho : m_1 m_2 \dots \mapsto (m_1, d_1)(m_2, d_2) \dots$$

be the *annotation map*. Note that since D is a DFA ρ is a bijection between paths in M and paths in $M \oslash_{mc} D$.

Lemma 20. *Let M be a Σ -labelled MC, D a DFA over $\Sigma \times Q_M$. Then for every measurable $X \subset (Q_M)^\omega$, $\text{Pr}_M(X) = \text{Pr}_{M \oslash_{mc} D}(\rho(X))$.*

For the proof it is enough to consider X of the form $\text{cone}(x)$.

Example 21. Let R be a DFA over Σ with final states F_R and $U = U_\Sigma$ the uniform Markov chain. The lemma says that Pr_U of the set of paths in need of R -repair equals the probability in $\text{Pr}_{U \oslash_{mc} R}$ of the set of paths that reach a state of the form $\Sigma \times F_R$.

Definition 22. A Markov decision process is a tuple $((V, E), (V_{dec}, V_{rand}), \mu_\iota, \mu)$ where

- (V, E) is a directed graph, V_{dec}, V_{rand} partition V ,
- $\mu : V_{rand} \rightarrow \mathbf{dbn}(V_{dec})$ is the edge distribution,
- $\mu_\iota \in \mathbf{dbn}(V_{rand})$ is the initial distribution,
- for $u \in V_{rand}, (u, v) \in E$ iff $\mu(u)(v) > 0$,

- for $v \in V$, $E(v)$ (the out-going edges from v) is non-empty.

We say that the vertices V_{dec} belong to the **decider**, while the vertices V_{rand} belong to the **randomizer**. A play is a path in (V, E) .

We think of a labelled MDP just as a labelled MC with the addition that decider's edges are labelled by elements from a set **Act**. Formally:

Definition 23 (Labelled MDP). An MDP is (Σ, Act) -labelled if

- for each $v \in V_{rand}$ the edges from v are in bijection with Σ , and
- for each $v \in V_{dec}$ each edge from v is labelled by an element of **Act**.

Note (identification) Suppose $|E(u)| = 1$ for all $u \in V_{dec}$. Then a (Σ, Act) -labelled MDP can be naturally viewed as Σ -labelled MC as well as a streaming transducer with input alphabet Σ and output values taken from **Act**.⁶ We call this *identification* and write things like “this MDP is the same, modulo the identification, as that MC”.

Definition 24 (Strategy in an MDP). A strategy σ for the decider is a function $\sigma: V^* \cdot V_{dec} \rightarrow V$ such that for all $w \in V^*$ and all $v \in V_{dec}$ we have $\sigma(w \cdot v) \in E(v)$. A memoryless strategy for the decider is independent of the history and depends only on the current state, and can be described as a function $\sigma: V_{dec} \rightarrow V$. A finite-state strategy for the decider is one induced by a DFA (Q, δ, ι) over input alphabet V and output function $\theta: V \times Q \rightarrow V$ as follows: $\sigma(wv) := \theta(v, \delta(\iota, w))$.

As usual, applying a strategy s to an MDP G results in a MC, which we write $G[s]$.

Definition 25. Let s be a finite-state strategy in G . Write \mathfrak{T}_s for the streaming transducer associated with $G[s]$. Note that \mathfrak{T}_s has input alphabet Σ and outputs elements from **Act**.

We now define a certain interleaving of a MC and an NFA yielding an MDP. The idea is that the MC determines the allowed moves of the randomizer while the NFA determines the allowed moves of the decider.

Definition 26 ($M \oslash_{mdp} N$). Suppose M is a Σ -labelled Markov chain and N is an NFA over alphabet $\Sigma \times \text{Act}$. Define a (Σ, Act) -labelled MDP $M \oslash_{mdp} N$ as follows:

- the randomizer's nodes are $Q_M \times Q_N$,
- the decider's nodes are $Q_M \times \Sigma \times Q_N$;
- if in M there is an edge from m to m' with label σ and probability x , then for all n there is an edge in $M \oslash_{mdp} N$ from (m, n) to (m', σ, n) with label σ and probability x ;

⁶ Later **Act** will be a set of strings output by a streaming transducer.

- if in N there is a transition from n to n' labelled $\sigma \in \Sigma$ and $a \in \text{Act}$, then for all m there is an edge from (m, σ, n) to (m, n') labelled a ;
- the initial distribution sends (m, n) to $\iota_M(m)$ if n is the initial state of N , and to 0 otherwise.

Note. In case D is a DFA then $M \oslash_{mdp} D$ (with the `Act`-labelling removed) is, modulo identification, the Σ -labelled MC $M \oslash_{mc} D$.

Lemma 27. *If s is a finite-state strategy then the MC $(M \oslash_{mdp} N)[s]$ is, modulo identification, the same as $M \oslash_{mc} D$ for some DFA D .*

An objective Φ for a game graph is a subset of plays. We consider two types of objectives, reachability and safety objectives. Given a set $X \subset V$ of nodes, the reachability objective `reach`(X) requires that some vertex in X be visited, and dually, the safety objective `safe`(X) requires that only vertices in X be visited. *Solving an MDP for objective Φ* means finding a strategy s such that, amongst all possible strategies, the probability in the chain $G[s]$ of Φ is maximised. We call this maximal probability the *value* of the MDP for objective Φ . The following lemma is a slight variation on the standard problem of solving MDPs with reachability objectives.

Lemma 28. *There is a PTIME algorithm that computes the value (and strategy) of an MDP whose objective is a boolean combination of reachability objectives.*

3.2 Proof of Theorem 14

From DFAs R and T and non-negative integer α we construct an MDP $G_{R,T,\alpha}$ and objectives `reach`(Ob_R) and `safe`(Ob_S) such that the following two quantities are equal: 1) the maximum probability over all strategies of `safe`(Ob_S) conditioned on `reach`(Ob_R); 2) the largest $\eta \in [0, 1]$ for which there exists an (η, α) -streaming transducer sending R to T . We now provide the construction (in I), then show how to compute the value (in II), and finally prove that the value is equal to the required conditional probability (in III).

I. Construction of MDP $G_{R,T,\alpha}$. The MDP is constructed in three steps:

Step 1. From the DFA $R = (Q_R, \Sigma, \delta_R, q_{0R}, F_R)$ construct the Σ -labelled Markov chain $UR := U_\Sigma \oslash_{mc} R$, as in Example 21. Its state set is $\Sigma \times Q_R$.

Step 2. From the DFA $T = (Q_T, \Sigma, \delta_T, q_{0T}, F_T)$ and non-negative integer α construct an NFA (without final states) T_α over alphabet $\Sigma \times \Sigma^*$ that simulates the possible repairs of the input string. It does this by storing the allowed number of edit operations left. The non-determinism will correspond to Decider's choices in the MDP. First we need some notation. Write `BEST-STR`(q, q', σ) for some fixed string w (say the length-lexicographically least) amongst those for which `ED`(w, σ) is minimal with the property that $\delta_T(q, w) = q'$. Now, define T_α as follows:

- the states are $Q_T \times \{\perp, 0, 1, \dots, \alpha\}$ (here \perp means we have failed to repair the input string);
- the initial state is (q_{0T}, α) (meaning initially there are α edit operations available);
- on input (σ, σ) there is a transition from (q, \perp) to $(\delta_T(q, \sigma), \perp)$, (ie. once we have failed to repair, just copy the input to the output);
- on input $(\sigma, \text{BEST-STR}(q, q', \sigma))$ there is a transition from (q, n) to (q', m) where

$$m = n - \text{ED}(\text{BEST-STR}(q, q', \sigma), \sigma)$$

if this quantity is non-negative, and otherwise $m = \perp$.

Step 3. Define the Σ -labelled MDP $G_{R,T,\alpha}$ as \oslash_{mdp} -product of the Markov chain UR and NFA T_α .

Notation. Every node in V_{rand} is of the form $(q, t, n) \in Q_{UR} \times Q_T \times \{\perp, 0, 1, \dots, \alpha\}$. Every node in V_{dec} is of the form $(q, \sigma', t, n) \in Q_{UR} \times \Sigma \times Q_T \times \{\perp, 0, 1, \dots, \alpha\}$. The second component of the element $q \in Q_{UR} := \Sigma \times Q_R$ is called the Q_R -component of (q, t, n) and of (q, σ', t, n) .

Objectives. We introduce two objectives. Let \mathbf{Ob}_R be the set of states of $G_{R,T,\alpha}$ whose Q_R -component is in F_R . Let \mathbf{Ob}_S be the set of states of $G_{R,T,\alpha}$ such that if the Q_R -component is in F_R then both $t \in F_T$ and $n \neq \perp$. The two objectives are $\mathbf{reach}(\mathbf{Ob}_R)$ and $\mathbf{safe}(\mathbf{Ob}_S)$.

II. Computing the Value of $G_{R,T,\alpha}$. Given R, T and α , the value η_* of $G_{R,T,\alpha}$ is defined as the maximum, over all strategies s , of the conditional probability,

$$\Pr_{G_{R,T,\alpha}[s]} (\mathbf{safe}(\mathbf{Ob}_S) \mid \mathbf{reach}(\mathbf{Ob}_R)) = \frac{\Pr_{G_{R,T,\alpha}[s]} (\mathbf{safe}(\mathbf{Ob}_S) \cap \mathbf{reach}(\mathbf{Ob}_R))}{\Pr_{G_{R,T,\alpha}[s]} (\mathbf{reach}(\mathbf{Ob}_R))}$$

Proposition 29. *The value of the mdp $G_{R,T,\alpha}$ is computable in PTIME and can be realised by a memoryless strategy.*

For the proof observe that the value of $\mathbf{reach}(\mathbf{Ob}_R)$ is independent of the strategy s chosen by the decider. This is so because s does not have any effect on the Q_R -component of the state of $G_{R,T,\alpha}$. Thus the value of $\mathbf{reach}(\mathbf{Ob}_R)$ can be easily calculated (fix any memoryless strategy and apply Lemma 18). So, we just need to find the value of the objective $\mathbf{safe}(\mathbf{Ob}_S) \cap \mathbf{reach}(\mathbf{Ob}_R)$. By Lemma 28 this can be computed, and the required strategy is memoryless.

III. Existence of an Optimal Streaming Transducer. Fix R, T and α , and let η_* be the value of the MDP $G_{R,T,\alpha}$ for the property $\mathbf{safe}(\mathbf{Ob}_S)$ conditioned on $\mathbf{reach}(\mathbf{Ob}_R)$. In this section Proposition 31 immediately implies what we want, namely: 1) there is an (η_*, α) -streaming transducer from R to T ; 2) there is no (η, α) -streaming transducer from R to T with $\eta > \eta_*$.

Lemma 30. *Let s be a finite-state strategy for $G_{R,T,\alpha}$ and \mathfrak{T}_s the corresponding streaming transducer. Then the probability in MC $G_{R,T,\alpha}[s]$ of $\mathbf{safe}(\mathbf{Ob}_S) \cap \mathbf{reach}(\mathbf{Ob}_R)$ divided by the probability of $\mathbf{reach}(\mathbf{Ob}_R)$ is equal to the probability that a string in need of R -repair is $\langle R, T \rangle$ -repaired by \mathfrak{T}_s within α .*

Proof. By Lemma 27 $G_{R,T,\alpha}[\mathbf{s}]$ is a MC of the form $UR \oslash_{mc} D$ for some DFA D . Thus the probability of $\text{reach}(\mathbf{Ob}_R)$ in $G_{R,T,\alpha}[\mathbf{s}]$ is equal to the probability in $UR \oslash_{mc} D$ of reaching a state whose Q_R -component is in F_R . Note that the annotation map ρ preserves the property that a path reaches a state whose Q_R -component is in F_R . By Lemma 20 the latter is equal to the probability in UR that a path reaches a state whose Q_R -component is in F_R . By Example 21 this is equal to the probability that that an infinite string is in need of repair. Similarly it can be shown that the probability of $\text{safe}(\mathbf{Ob}_S)$ in $G_{R,T,\alpha}[\mathbf{s}]$ is equal to the probability in that an infinite string is $\langle R, T \rangle$ -repairable by $\mathfrak{T}_{\mathbf{s}}$ within α ; and the same for the intersection. \square

Proposition 31. 1. From a memoryless strategy \mathbf{s} in $G_{R,T,\alpha}$ with probability (of the conditional objective) η one can construct an (η, α) -streaming transducer from R to T .
 2. From an (η, α) -streaming transducer \mathfrak{T} from R to T one can construct a strategy $\mathbf{s}_{\mathfrak{T}}$ in $G_{R,T,\alpha}$ with value $\geq \eta$.

Proof. The first item is immediate from Lemma 30.

For the second, a transducer \mathfrak{T} gives rise to the following strategy $\mathbf{s}_{\mathfrak{T}}$: suppose $\rho \in V^*$ is a play ending in $(q, \sigma, t, n) \in V_{dec}$ where q is a state of UR and t of T_α . Let $\text{in}(\rho)$ be the input (ie. the letters that Randomizer has chosen) and note that it ends in σ . The strategy $\mathbf{s}_{\mathfrak{T}}$ sends ρ to node $(q, t', n') \in V_{rand}$, where $t' = \delta_T(q_{0T}, \mathfrak{T}(\text{in}(\rho)))$ where δ_T is the transition function and q_{0T} the initial state of the DFA for T . Note that $\mathbf{s}_{\mathfrak{T}}$ is a finite-state strategy. So let \mathfrak{T}' be the transducer associated with strategy $\mathbf{s}_{\mathfrak{T}}$ and apply Lemma 30. Then the probability in $G_{R,T,\alpha}[\mathbf{s}_{\mathfrak{T}}]$ of $\text{safe}(\mathbf{Ob}_S) \cap \text{reach}(\mathbf{Ob}_R)$ divided by the probability of $\text{reach}(\mathbf{Ob}_R)$ is equal to the probability that a string in need of R -repair is $\langle R, T \rangle$ -repaired by \mathfrak{T}' within α . It is required to show that this latter probability is at least η . For this it is sufficient to show that if a string is repaired by \mathfrak{T} then it is repaired by \mathfrak{T}' . But this is the case because although both \mathfrak{T} and \mathfrak{T}' suggest the same next state for a given input string, say from (q, σ, t, n) to (q, t', n') , they possibly differ on the output string, say u and v . In particular, $\text{ED}(\sigma, u) \geq \text{ED}(\sigma, v) := \text{BEST-STR}(t, t', \sigma)$ by the definition of $G_{R,T,\alpha}$. \square

References

1. Rajeev Alur, Loris D’Antoni, Jyotirmoy V. Deshmukh, Mukund Raghavan, and Yifei Yuan. Regular functions, cost register automata, and generalized min-cost problems. *CoRR*, abs/1111.0670, 2011.
2. Michael Benedikt, Gabriele Puppis, and Cristian Riveros. Regular repair of specifications. In *LICS*, pages 335–344, 2011.
3. Robert A. Wagner. Order-n correction for regular languages. *Commun. ACM*, 17(5):265–268, may 1974.

Parameterized Model Checking of Token-Passing Systems

Benjamin Aminof¹, Swen Jacobs², Ayrat Khalimov², Sasha Rubin^{1,3 *}

¹IST Austria (first.last@ist.ac.at), ²TU Graz (first.last@iaik.tugraz.at), ³TU Wien

Abstract. We revisit the parameterized model checking problem for token-passing systems and specifications in indexed $\text{CTL}^*\backslash X$. Emerson and Namjoshi (1995, 2003) have shown that parameterized model checking of indexed $\text{CTL}^*\backslash X$ in uni-directional token rings can be reduced to checking rings up to some *cutoff* size. Clarke et al. (2004) have shown a similar result for general topologies and indexed $\text{LTL}\backslash X$, provided processes cannot choose the directions for sending or receiving the token.

We unify and substantially extend these results by systematically exploring fragments of indexed $\text{CTL}^*\backslash X$ with respect to general topologies. For each fragment we establish whether a cutoff exists, and for some concrete topologies, such as rings, cliques and stars, we infer small cutoffs. Finally, we show that the problem becomes undecidable, and thus no cutoffs exist, if processes are allowed to choose the directions in which they send or from which they receive the token.

1 Introduction

As executions of programs and protocols are increasingly distributed over multiple CPU cores or even physically separated computers, correctness of concurrent systems is one of the primary challenges of formal methods today. Many concurrent systems consist of an arbitrary number of identical processes running in parallel. The parameterized model checking problem (PMCP) for concurrent systems is to decide if a given temporal logic specification holds irrespective of the number of participating processes.

The PMCP is undecidable in many cases. For example, it is undecidable already for safety specifications and finite-state processes communicating by passing a binary-valued token around a uni-directional ring [14,7]. However, decidability may be regained by restricting the communication primitives, the topologies under consideration (i.e., the underlying graph describing the communication paths between the processes), or the specification language. In particular, previous results have shown that parameterized model checking can sometimes be reduced to model checking a finite number of instances of the system, up to some *cutoff* size.

* This work was supported by the Austrian Science Fund through grant P23499-N23 and through the RiSE network (S11403, S11405, S11406, S11407-N23); ERC Starting Grant (279307: Graph Games); Vienna Science and Technology Fund (WWTF) grants PROSEED, ICT12-059, and VRG11-005.

For token-passing systems (TPSs) with uni-directional ring topologies, such cutoffs are known for specifications in the prenex fragment of indexed CTL^* without the next-time operator ($\text{CTL}^*\backslash X$) [9,7]. For token-passing in general topologies, cutoffs are known for the prenex fragment of indexed $\text{LTL}\backslash X$, provided that processes are not allowed to choose the direction to which the token is sent or from which it is received [4]. In this paper we generalize these results and elucidate what they have in common.

Previous Results. In their seminal paper [7], Emerson and Namjoshi consider systems where the token does not carry messages, and specifications are in prenex indexed temporal logic — i.e., quantifiers \forall and \exists over processes appear in a block at the front of the formula. They use the important concept of a *cutoff* — a number c such that the PMCP for a given class of systems and specifications can be reduced to model checking systems with up to c processes. Clearly, if there is a cutoff for a given class and processes are finite state, then the PMCP for this class is decidable. Conversely, if the PMCP is undecidable, then there can be no cutoff for such systems.

For uni-directional rings, Emerson and Namjoshi provide cutoffs for formulas with a small number k of quantified index variables, and state that their proof method allows one to obtain cutoffs for other quantifier prefixes. In brief, cutoffs exist for the branching-time specification language prenex-indexed $\text{CTL}^*\backslash X$ and the highly regular topology of uni-directional rings.

Clarke et al. [4] consider the PMCP for token-passing systems arranged in general topologies. Their main result is that the PMCP for systems with arbitrary topologies and k -indexed $\text{LTL}\backslash X$ specifications (i.e., specifications with k quantifiers over processes) can be reduced to combining the results of model-checking finitely many topologies of size at most $2k$ [4, Theorem 4]. Their proof implies that, for each k , the PMCP for linear-time specifications in k -indexed $\text{LTL}\backslash X$ and general topologies has a cutoff.

Questions. Comparing these results, an obvious question is: are there cutoffs for branching time temporal logics and arbitrary topologies (see Table 1)? Clarke et al. already give a first answer [4, Corollary 3]. They prove that there is no cutoff for token-passing systems with arbitrary topologies and specifications from 2-indexed $\text{CTL}\backslash X$. However, their proof makes use of formulas with unbounded nesting-depth of path quantifiers. This lead us to the first question.

	Uni-Ring Topologies	Arbitrary Topologies
indexed $\text{LTL}\backslash X$	—	[4]
indexed $\text{CTL}^*\backslash X$	[7]	this paper

Table 1: Direction-Unaware TPSs.

	Bi-Ring Topologies	Arbitrary Topologies
indexed $\text{LTL}\backslash X$	[6]	this paper
indexed $\text{CTL}^*\backslash X$	this paper	this paper

Table 2: Direction-Aware TPSs.

Question 1. Is there a way to stratify k -indexed $\text{CTL}^*\backslash X$ such that for each level of the stratification there is a cutoff for systems with arbitrary topologies? In particular, does stratification by nesting-depth of path quantifiers do the trick?

Cutoffs for k -indexed temporal logic fragments immediately yield that for each k there is an algorithm (depending on k) for deciding the PMCP for k -indexed temporal logic. However, this does not imply that there is an algorithm that can compute the cutoff for a given k . In particular, it does not imply that PMCP for full prenex indexed temporal logic is decidable.

Question 2. For which topologies (rings, cliques, all?) can one conclude that the PMCP for the full prenex-indexed temporal logic is decidable?

Finally, an important implicit assumption in Clarke et al. [4] is that processes are not *direction aware*, i.e., they cannot sense or choose in which direction the token is sent, or from which direction it is received. In contrast, Emerson and Kahlon [6] show that cutoffs exist for certain direction-aware systems in bi-directional rings (see Section 8). We were thus motivated to understand to what extent existing results about cutoffs can be lifted to direction-aware systems, see Table 2.

Question 3. Do cutoffs exist for direction-aware systems on arbitrary topologies and k -indexed temporal logics (such as $\text{LTL}\backslash X$ and $\text{CTL}\backslash X$)?

Our contributions. In this paper, we answer the questions above, unifying and substantially extending the known cutoff results:

Answer to Question 1. Our main positive result (Theorem 7) states that for arbitrary parameterized topologies \mathbf{G} there is a cutoff for specifications in k -indexed $\text{CTL}_d^*\backslash X$ — the cutoff depends on \mathbf{G} , the number k of the process quantifiers, and the nesting depth d of path quantifiers. In particular, indexed $\text{LTL}\backslash X$ is included in the case $d = 1$, and so our result generalizes the results of Clarke et al. [4].

Answer to Question 2. We prove (Theorem 14) that there exist topologies for which the PMCP is undecidable for specifications in prenex-indexed $\text{CTL}\backslash X$ or $\text{LTL}\backslash X$. Note that this undecidability result does not contradict the existence of cutoffs (Theorem 7), since cutoffs may not be computable from k, d (see the note on decidability in Section 2.4). However, for certain topologies our positive result is constructive and we can compute cutoffs given k and d (Theorem 15). To illustrate, we show that rings have a cutoff of $2k$, cliques of $k+1$, and stars of $k+1$ (independent of d). In particular, PMCP is decidable for these topologies and specifications in prenex-indexed $\text{CTL}^*\backslash X$.

Answer to Question 3. The results just mentioned assume that processes are not direction-aware. Our main negative result (Theorem 18) states that if processes can control at least one of the directions (i.e., choose in which direction to send or from which direction to receive) then the PMCP for arbitrary topologies and k -indexed logic (even $\text{LTL}\backslash X$ and $\text{CTL}\backslash X$) is undecidable, and therefore does not have cutoffs. Moreover, if processes can control both in- and out-directions, then the PMCP is already undecidable for bi-directional rings and 1-indexed $\text{LTL}\backslash X$.

Technical contributions relative to previous work. Our main positive result (Theorem 7) generalizes proof techniques and ideas from previous results [7,4]. We observe that in both of these papers the main idea is to abstract a TPS by simulating the quantified processes exactly and simulating the movement of the token between these processes. The relevant information about the movement of the token is this: whether there is a direct edge, or a path (through unquantified processes) from one quantified process to another. This abstraction does not work for $\text{CTL}_d^* \setminus X$ and general topologies since the formula can express branching properties of the token movement. Our main observation is that the relevant information about the branching-possibilities of the token can be expressed in $\text{CTL}_d^* \setminus X$ over the topology itself. We develop a composition-like theorem, stating that if two topologies (with k distinguished vertices) are indistinguishable by $\text{CTL}_d^* \setminus X$ formulas, then the TPSs based on these topologies and an arbitrary process template P are indistinguishable by $\text{CTL}_d^* \setminus X$ (Theorem 9). The machinery involves a generalization of stuttering trace-equivalence [13], a notion of d -contraction that serves the same purpose as the connection topologies of Clarke et al. [4, Proposition 1], and also the main simulation idea of Emerson and Namjoshi [7, Theorem 2].

Our main negative result, undecidability of PMCP for direction-aware systems (Theorem 18), is proven by establishing a reduction from the non-halting problem for 2-counter machines (as is typical in this area [7,10]). Due to the lack of space, full proofs are omitted, and can be found in the full version [1].

2 Definitions and Existing Results

Let \mathbb{N} denote the set of positive integers. Let $[k]$ for $k \in \mathbb{N}$ denote the set $\{1, \dots, k\}$. The concatenation of strings u and w is written uw or $u \cdot w$.

Let AP denote a countably infinite set of *atomic propositions* or *atoms*. A *labeled transition system (LTS)* over AP is a tuple $(Q, Q_0, \Sigma, \delta, \lambda)$ where Q is the set of *states*, $Q_0 \subseteq Q$ are the *initial states*, Σ is the set of *transition labels* (also called *action labels*), $\delta \subseteq Q \times \Sigma \times Q$ is the *transition relation*, and $\lambda : Q \rightarrow 2^{\text{AP}}$ is the *state-labeling* and satisfies that $\lambda(q)$ is finite (for every $q \in Q$). Transitions $(q, \sigma, q') \in \delta$ may be written $q \xrightarrow{\sigma} q'$.

A *state-action path* of an LTS $(Q, Q_0, \Sigma, \delta, \lambda)$ is a finite sequence $q_0 \sigma_0 q_1 \sigma_1 \dots q_n \in (Q\Sigma)^* Q$ or an infinite sequence $q_0 \sigma_0 q_1 \sigma_1 \dots \in (Q\Sigma)^\omega$ such that $(q_i, \sigma_i, q_{i+1}) \in \delta$ (for all i). A *path* of an LTS is the projection $q_0 q_1 \dots$ of a state-action path onto states Q . An *action-labeled path* of an LTS is the projection $\sigma_0 \sigma_1 \dots$ of a state-action path onto transition labels Σ .

2.1 System Model (Direction-Unaware)

In this section we define the LTS P^G — it consists of replicated copies of a process P placed on the vertices of a graph G . Transitions in P^G are either internal (in which exactly one process moves) or synchronized (in which one

process sends the token to another along an edge of G). The token starts with the process that is at the initial vertex of G .

Fix a countably infinite set of (local) atomic propositions AP_{pr} (to be used by the states of the individual processes).

Process Template P . Let Σ_{int} denote a finite non-empty set of *internal-transition labels*. Define Σ_{pr} as the disjoint union $\Sigma_{\text{int}} \cup \{\text{rcv}, \text{snd}\}$ where rcv and snd are new symbols.

A *process template P* is a LTS $(Q, Q_0, \Sigma_{\text{pr}}, \delta, \lambda)$ over AP_{pr} such that:

- i) the state set Q is finite and can be partitioned into two non-empty sets, say $T \cup N$. States in T are said to *have the token*.
- ii) The initial state set is $Q_0 = \{\iota_t, \iota_n\}$ for some $\iota_t \in T, \iota_n \in N$.
- iii) Every transition $q \xrightarrow{\text{snd}} q'$ satisfies that q has the token and q' does not.
- iv) Every transition $q \xrightarrow{\text{rcv}} q'$ satisfies that q' has the token and q does not.
- v) Every transition $q \xrightarrow{a} q'$ with $a \in \Sigma_{\text{int}}$ satisfies that q has the token if and only if q' has the token.
- vi) The transition relation δ is total in the first coordinate: for every $q \in Q$ there exists $\sigma \in \Sigma_{\text{pr}}, q' \in Q$ such that $(q, \sigma, q') \in \delta$ (i.e., the process P is non-terminating).
- vii) Every infinite action-labeled path $a_0 a_1 \dots$ is in the set $(\Sigma_{\text{int}}^* \text{ snd } \Sigma_{\text{int}}^* \text{ rcv})^\omega \cup (\Sigma_{\text{int}}^* \text{ rcv } \Sigma_{\text{int}}^* \text{ snd})^\omega$ (i.e., snd and rcv actions alternate continually along every infinite action-labeled path of P).¹

The elements of Q are called *local states* and the transitions in δ are called *local transitions (of P)*. A local state q such that the only transitions are of the form $q \xrightarrow{\text{snd}} q'$ (for some q') is said to be *send-only*. A local state q such that the only transitions are of the form $q \xrightarrow{\text{rcv}} q'$ (for some q') is said to be *receive-only*.

Topology G . A *topology*² is a directed graph $G = (V, E, x)$ where $V = [k]$ for some $k \in \mathbb{N}$, vertex $x \in V$ is the *initial vertex*, $E \subseteq V \times V$, and $(v, v) \notin E$ for every $v \in V$. Vertices are called *process indices*.

We may also write $G = (V_G, E_G, x_G)$ if we need to disambiguate.

Token-Passing System P^G . Let $\text{AP}_{\text{sys}} := \text{AP}_{\text{pr}} \times \mathbb{N}$ be the *indexed atomic propositions*. For $(p, i) \in \text{AP}_{\text{sys}}$ we may also write p_i . Given a process template $P = (Q, Q_0, \Sigma_{\text{pr}}, \delta, \lambda)$ over AP_{pr} and a topology $G = (V, E, x)$, define the *token-passing system (TPS) P^G* as the finite LTS $(S, S_0, \Sigma_{\text{int}} \cup \{\text{tok}\}, \Delta, \Lambda)$ over atomic propositions $\text{AP}_{\text{sys}} := \text{AP}_{\text{pr}} \times \mathbb{N}$, where:

- The set S of *global states* is Q^V , i.e., all functions from V to Q . If $s \in Q^V$ is a global state then $s(i)$ denotes the local state of the process with index i .³
- The set of *global initial states* S_0 consists of the unique global state $s_0 \in Q_0^V$ such that only $s_0(x)$ has the token.

¹ This restriction was introduced by Emerson and Namjoshi in [7]. Our positive results that cutoffs exist also hold for a more liberal restriction (see Section 7).

² There does not seem to be standard terminology for this object. The paper [4] uses ‘network graph’ and reserves ‘topology’ for a set of network graphs.

³ In [7,4] a global state is defined to be a vector of local states, which is a notational variation of the functional notation we prefer.

- The labeling $\Lambda(s) \subset AP_{sys}$ for $s \in S$ is defined as follows: $p_i \in \Lambda(s)$ if and only if $p \in \lambda(s(i))$, for $p \in AP_{pr}$ and $i \in V$.
- The *global transition relation* Δ is defined to consist of the set of all internal transitions and synchronous transitions:
 - An *internal transition* is an element (s, a, s') of $S \times \Sigma_{int} \times S$ for which there exists a process index $v \in V$ such that
 - i) $s(v) \xrightarrow{a} s'(v)$ is a local transition of P , and
 - ii) for all $w \in V \setminus \{v\}$, $s(w) = s'(w)$.
 - A *token-passing transition* is an element (s, tok, s') of $S \times \{tok\} \times S$ for which there exist process indices $v, w \in V$ such that $(v, w) \in E$ and
 - i) $s(v) \xrightarrow{snd} s'(v)$ is a local transition of P ,
 - ii) $s(w) \xrightarrow{rcv} s'(w)$ is a local transition of P , and
 - iii) for every $u \in V \setminus \{v, w\}$, $s'(u) = s(u)$.

In words, the system P^G can be thought of the asynchronous parallel composition of P over topology G . The token starts with process x . At each time step either exactly one process makes an internal transition, or exactly two processes synchronize when one process sends the token to another along an edge of G .

2.2 System Model (Direction-Aware)

Inspired by direction-awareness in the work of Emerson and Kahlon [6], we extend the definition of TPS to include additional labels on edges, called *directions*. The idea is that processes can restrict which directions are used when they send or receive the token. Fix finite non-empty disjoint sets Dir_{snd} of *sending directions* and Dir_{rcv} of *receiving directions*. A *direction-aware token-passing system* is a TPS with the following modifications.

Direction-aware Topology. A *direction-aware topology* is a topology $G = (V, E, x)$ with labeling functions $dir_{rcv} : E \rightarrow Dir_{rcv}$, $dir_{snd} : E \rightarrow Dir_{snd}$.

Direction-aware Process Template. In direction-aware systems, process templates use transition labels from $\Sigma_{pr} := \Sigma_{int} \cup Dir_{snd} \cup Dir_{rcv}$. The definition of a *direction-aware process template* is like that in Section 2.1, except that in item iii) snd is replaced by $d \in Dir_{snd}$, in iv) rcv is replaced by $d \in Dir_{rcv}$, and in vii) snd is replaced by Dir_{snd} and rcv by Dir_{rcv} .

Direction-aware Token Passing System. Fix Dir_{snd} and Dir_{rcv} , let G be a direction-aware topology and P a direction-aware process template. Define the *direction-aware token-passing system* P^G as in Section 2.1, except that token-passing transitions are now direction-aware:

Direction-aware token-passing transitions are elements (s, tok, s') of $S \times \{tok\} \times S$ for which there exist process indices $v, w \in V$ with $(v, w) \in E$, $dir_{snd}(v, w) = d$, and $dir_{rcv}(v, w) = e$, such that:

- i) $s(v) \xrightarrow{d} s'(v)$ is a local transition of P .
- ii) $s(w) \xrightarrow{e} s'(w)$ is a local transition of P .
- iii) For every $u \in V \setminus \{v, w\}$, $s'(u) = s(u)$.

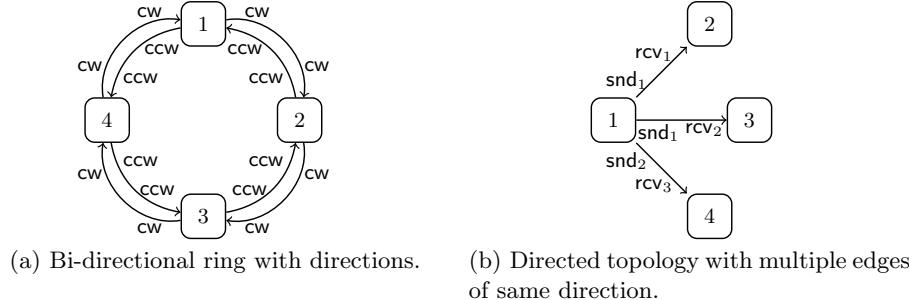


Fig. 1: Direction-aware Topologies.

Notations \mathcal{P}_u , \mathcal{P}_{snd} , \mathcal{P}_{rcv} , $\mathcal{P}_{\text{sndrcv}}$. Let \mathcal{P}_u denote the set of all process templates for which $|\text{Dir}_{\text{snd}}| = |\text{Dir}_{\text{rcv}}| = 1$. In this case P^G degenerates to a direction-unaware TPS as defined in Section 2.1. If we require $|\text{Dir}_{\text{rcv}}| = 1$, then processes cannot choose from which directions to receive the token, but possibly in which direction to send it. Denote the set of all such process templates by \mathcal{P}_{snd} . Similarly define \mathcal{P}_{rcv} to be all process templates where $|\text{Dir}_{\text{snd}}| = 1$ — processes cannot choose where to send the token, but possibly from which direction to receive it. Finally, let $\mathcal{P}_{\text{sndrcv}}$ be the set of all direction-aware process templates.

Examples. Figure 1(a) shows a bi-directional ring with directions cw (clockwise) and ccw (counterclockwise). Every edge e is labeled with an outgoing direction $\text{dir}_{\text{snd}}(e)$ and an incoming direction $\text{dir}_{\text{rcv}}(e)$.⁴ Using these directions, a process that has the token can choose whether he wants to send it in direction cw or ccw. Depending on its local state, a process waiting for the token can also choose to receive it only from direction cw or ccw.

Figure 1(b) depicts a topology in which process 1 can choose between two outgoing directions. If it sends the token in direction snd_1 , it may be received by either process 2 or 3. If however process 2 blocks receiving from direction rcv_1 , the token can only be received by process 3. If 3 additionally blocks receiving from rcv_2 , then this token-passing transition is disabled.

2.3 Indexed Temporal Logics

Indexed temporal logics (ITL) were introduced in [3,8,7] to model specifications of certain distributed systems. Subsequently one finds a number of variations of indexed temporal-logics in the literature (linear vs. branching, restrictions on the quantification). Thus we introduce Indexed-CTL* which has these variations (and those in [4]) as syntactic fragments.

Syntactic Fragments of CTL*, and \equiv_{TL} . We assume the reader is familiar with the syntax and semantics of CTL*, for a reminder see [2]. For $d \in \mathbb{N}$ let $\text{CTL}_d^* \setminus X$ denote the syntactic fragment of $\text{CTL}^* \setminus X$ in which the nesting-depth of path quantifiers is at most d (for a formal definition see [13, Section 4]).

⁴ For notational simplicity, we denote both outgoing direction snd_{cw} and incoming direction rcv_{cw} by cw, and similarly for ccw.

Let TL denote a temporal logic (in this paper these are fragments of $\text{CTL}^*\setminus\mathsf{X}$). For temporal logic TL and LTSs M and N , write $M \equiv_{\text{TL}} N$ to mean that for every formula $\phi \in \text{TL}$, $M \models \phi$ if and only if $N \models \phi$.

Indexed-CTL^{*}. Fix an infinite set $\text{Vars} = \{x, y, z, \dots\}$ of index variables, i.e., variables with values from \mathbb{N} . These variables refer to vertices in the topology.

Syntax. The *Indexed-CTL^{*}* formulas over variable set Vars and atomic propositions AP are formed by adding the following rules to the syntax of CTL^* over atomic propositions $\text{AP} \times \text{Vars}$. We write p_x instead of $(p, x) \in \text{AP} \times \text{Vars}$.

If ϕ is an indexed-CTL^{*} state (resp. path) formula and $x, y \in \text{Vars}, Y \subset \text{Vars}$ then the following are also indexed-CTL^{*} state (resp. path) formulas:

- $\forall x.\phi$ (meaning that for all vertices in the topology ϕ should hold),
- $\forall x.x \notin Y \rightarrow \phi$ (meaning that ϕ holds for all vertices that are different to those that are designated by variables in Y),
- $\forall x.x \in Y \rightarrow \phi$,
- $\forall x.x \in E(y) \rightarrow \phi$ (meaning that ϕ holds for all vertices to which there is an edge from the vertex designated by the variable y), and
- $\forall x.x \notin E(y) \rightarrow \phi$.

Shorthand. As usual, write $\exists x.\phi$ as shorthand for $\neg\forall x.\neg\phi$, write $\forall x \notin Y.\phi$ as shorthand for $\forall x.x \notin Y \rightarrow \phi$, and $\forall x \in E(y).\phi$ as shorthand for $\forall x.x \in E(y) \rightarrow \phi$. All these prefixes $\exists x, \forall x, \exists x \in E(y), \forall x \in E(y), \exists x \in Y, \dots$ are called *index quantifiers*. We use Qx to denote an index quantifier.

Semantics. Indexed temporal logic is interpreted over a system instance P^G (with P a process template and G a topology). The formal *semantics* are in the full version of the paper [1]. Here we give some examples. The formula $\forall i. \mathsf{EF} p_i$ states that for every process there exists a path such that eventually that process is in a state that satisfies atom p . The formula $\mathsf{EF} \forall i.p_i$ states that there is a path such that eventually all processes satisfy atom p simultaneously. We now define the central fragment of ITL which includes the former example and not the latter.

Prenex indexed TL and $\{\forall, \exists\}^k\text{-TL}$. *Prenex indexed temporal-logic* is a syntactic fragment of indexed temporal-logic in which all quantifiers are at the front of the formula, e.g., prenex-indexed $\text{LTL}\setminus\mathsf{X}$ consists of formulas of the form $(Q_1 x_1) \dots (Q_k x_k) \varphi$ where φ is an $\text{LTL}\setminus\mathsf{X}$ formula over atoms $\text{AP} \times \{x_1, \dots, x_k\}$, and the $Q_i x_i$ s are index quantifiers. Such formulas with k quantifiers will be referred to as *k-indexed*, collectively written $\{\forall, \exists\}^k\text{-TL}$. The union of $\{\forall, \exists\}^k\text{-TL}$ for $k \in \mathbb{N}$ is written $\{\forall, \exists\}^*\text{-TL}$ and called (full) prenex indexed TL .

2.4 Parameterized Model Checking Problem, Cutoffs, Decidability

A *parameterized topology* \mathbf{G} is a countable set of topologies. E.g., the set of unidirectional rings with all possible initial vertices is a parameterized topology.

PMCP_G(-, -). The *parameterized model checking problem* (PMCP) for parameterized topology \mathbf{G} , processes from \mathcal{P} , and parameterized specifications from

\mathcal{F} , written $\text{PMCP}_{\mathbf{G}}(\mathcal{P}, \mathcal{F})$, is the set of pairs $(\varphi, P) \in \mathcal{F} \times \mathcal{P}$ such that for all $G \in \mathbf{G}$, $P^G \models \varphi$.

A solution to $\text{PMCP}_{\mathbf{G}}(\mathcal{P}, \mathcal{F})$ is an algorithm that, given a formula $\varphi \in \mathcal{F}$ and a process template $P \in \mathcal{P}$ as input, outputs 'Yes' if for all $G \in \mathbf{G}$, $P^G \models \varphi$, and 'No' otherwise.

Cutoff. A *cutoff* for $\text{PMCP}_{\mathbf{G}}(\mathcal{P}, \mathcal{F})$ is a natural number c such that for every $P \in \mathcal{P}$ and $\varphi \in \mathcal{F}$, the following are equivalent:⁵

- $P^G \models \varphi$ for all $G \in \mathbf{G}$ with $|V_G| \leq c$;
- $P^G \models \varphi$ for all $G \in \mathbf{G}$.

Thus $\text{PMCP}_{\mathbf{G}}(\mathcal{P}, \mathcal{F})$ does not have a cutoff iff for every $c \in \mathbb{N}$ there exists $P \in \mathcal{P}$ and $\varphi \in \mathcal{F}$ such that $P^G \models \varphi$ for all $G \in \mathbf{G}$ with $|V_G| \leq c$, and there exists $G \in \mathbf{G}$ such that $P^G \not\models \varphi$.

Observation 1. If $\text{PMCP}_{\mathbf{G}}(\mathcal{P}, \mathcal{F})$ has a cutoff, then $\text{PMCP}_{\mathbf{G}}(\mathcal{P}, \mathcal{F})$ is decidable.

Indeed: if c is a cutoff, let G_1, \dots, G_n be all topologies G in \mathbf{G} such that $|V_G| \leq c$. The algorithm that solves PMCP takes P, φ as input and checks whether or not $P^{G_i} \models \varphi$ for all $1 \leq i \leq n$.

Note About Decidability. The following statements are not, a priori, equivalent (for given parameterized topology \mathbf{G} and process templates \mathcal{P}):

- For every $k \in \mathbb{N}$, $\text{PMCP}_{\mathbf{G}}(\mathcal{P}, \{\forall, \exists\}^k\text{-TL})$ is decidable.
- $\text{PMCP}_{\mathbf{G}}(\mathcal{P}, \{\forall, \exists\}^*\text{-TL})$ is decidable.

The first item says that for every k there exists an algorithm A_k that solves the PMCP for k -indexed TL. This does not imply the second item, which says that there exists a single algorithm that solves the PMCP for $\bigcup_{k \in \mathbb{N}} \{\forall, \exists\}^k\text{-TL}$. If the function $k \mapsto A_k$ is also computable (e.g., Theorem 15) then indeed the second item follows: given P, φ , extract the size k of the prenex block of φ , compute a description of A_k , and run A_k on P, φ .

For instance, the result of Clarke et al. (that there are cutoffs for k -index LTL\X and arbitrary topologies) does not imply that the PMCP for $\{\forall, \exists\}^*\text{-LTL}\setminus\mathsf{X}$ and arbitrary topologies is decidable. Aware of this fact, the authors state (after Theorem 4) "Note that the theorem does not provide us with an effective means to find the reduction [i.e. algorithm]...".

In fact, we prove (Theorem 14) that there is some parameterized topology such that PMCP is undecidable for prenex-indexed LTL\X.

Existing Results. We restate the known results using our terminology.

A *uni-directional ring* $G = (V, E, x)$ is a topology with $V = [n]$ for some $n \in \mathbb{N}$, there are edges $(i, i+1)$ for $1 \leq i \leq n$ (arithmetic is modulo n), and $x \in V$. Let \mathbf{R} be the parameterized topology consisting of all uni-directional rings.

Theorem 2 (Implicit in [7]). For every $k \in \mathbb{N}$, there is a cutoff for the problem $\text{PMCP}_{\mathbf{R}}(\mathcal{P}_u, \{\forall, \exists\}^k\text{-CTL}^*\setminus\mathsf{X})$.⁶

⁵ Of course the first item always follows from the second item.

⁶ The paper explicitly contains the result that 4 is a cutoff for $\forall\forall\text{-CTL}^*\setminus\mathsf{X}$ on rings. However the proof ideas apply to get the stated theorem.

Although Clarke et al. [4] do not explicitly state the following theorem, it follows from their proof technique, which we generalise in Section 3.

Theorem 3 (Implicit in [4]). *For every parameterized topology \mathbf{G} , and every $k \in \mathbb{N}$, the problem $\text{PMCP}_{\mathbf{G}}(\mathcal{P}_u, \{\forall, \exists\}^k\text{-}\mathit{TL}\backslash X)$ has a cutoff. A corollary [12, Corollary 2] states that $2k$ is a cutoff if \mathbf{G} is taken to be \mathbf{R}* ⁷.

Theorem 4 ([4, Corollary 3]). *There exists a parameterized topology \mathbf{G} and process $P \in \mathcal{P}_u$ such that the problem $\text{PMCP}_{\mathbf{G}}(\{P\}, \{\exists\}^2\text{-}\mathit{CTL}\backslash X)$ does not have a cutoff.*

The proof of this theorem defines \mathbf{G} , process P , and for every $c \in \mathbb{N}$ a formula φ_c , such that if $G \in \mathbf{G}$ then $P^G \models \varphi_c$ if and only if $|V_G| \leq c$. The formula φ_c is in 2-indexed $\mathit{CTL}\backslash X$ and has nesting depth of path quantifiers equal to c .

3 Method for Proving Existence of Cutoffs

We give a method for proving cutoffs for direction-*unaware* TPSs that will be used to prove Theorem 7. The method abstracts from the ideas of Clarke et al. [4].

In a k -indexed TL formula $Q_1x_1 \dots Q_kx_k. \varphi$, every valuation of the variables x_1, \dots, x_k designates k nodes of the underlying topology G , say $\bar{g} = g_1, \dots, g_k$. The formula φ can only talk about (the processes in) \bar{g} . In order to prove that the PMCP has a cutoff, it is sufficient (as the proof of Theorem 5 will demonstrate) to find conditions on two topologies G, G' and \bar{g}, \bar{g}' that allow one to conclude that P^G and $P^{G'}$ are indistinguishable with respect to φ .

We define two abstractions for a given TPS P^G . The first abstraction simulates P^G , keeping track only of the local states of processes indexed by \bar{g} . We call it the *projection* of P^G onto \bar{g} .⁸ The second abstraction only simulates the movement of the token in G , restricted to \bar{g} . We call it the *graph LTS* of G and \bar{g} .

Notation. Let \bar{g} denote a tuple (g_1, \dots, g_k) of *distinct* elements of V_G , and \bar{g}' a k -tuple of distinct elements of $V_{G'}$. Write $v \in \bar{g}$ if $v = g_i$ for some i .

The projection $P^G|\bar{g}$. Informally, the *projection* of P^G onto a tuple of process indices \bar{g} is the LTS P^G and a new labeling that, for every $g_i \in \bar{g}$, replaces the indexed atom p_{g_i} by the atom $p@i$; all other atoms are removed. Thus $p@i$ means that the atom $p \in \mathsf{AP}_{\text{pr}}$ holds in the process with index g_i . In other words, process indices are replaced by their *positions* in \bar{g} .

Formally, fix process P , topology G , and k -tuple \bar{g} over V_G . Define the *projection of P^G onto \bar{g}* as the LTS $(S, S_0, \Sigma_{\text{int}} \cup \{\text{tok}\}, \Delta, \Lambda)$ over atomic propositions $\{p@i : p \in \mathsf{AP}_{\text{pr}}, i \in [k]\}$, where for all $s \in S$ the labeling $L(s)$ is defined as follows: $L(s) := \{p@i : p_{g_i} \in \Lambda(s), i \in [k]\}$.

⁷ There is an error in [12, Corollary 2]: $2k$ is the cutoff only for formula with no quantifier alternations. See Remark 17 in Section 5.

⁸ Emerson and Namjoshi [7, Section 2.1] define the related notion ‘‘LTS projection’’.

The graph LTS $G|\bar{g}$. Informally, $G|\bar{g}$ is an LTS where states are nodes of the graph G , and transitions are edges of G . The restriction to \bar{g} is modeled by labeling a state with the position of the corresponding node in \bar{g} .

Let $G = (V, E, x)$ be a topology, and let \bar{g} be a k -tuple over V_G . Define the *graph LTS* $G|g$ as the LTS $(Q, Q_0, \Sigma, \Delta, \Lambda)$ over atomic propositions $\{1, \dots, k\}$, with state set $Q := V$, initial state set $Q_0 := \{x\}$, action set $\Sigma = \{\mathbf{a}\}$, transition relation with $(v, \mathbf{a}, w) \in \Delta$ iff $(v, w) \in E$, and labeling $\Lambda(v) := \{i\}$ if $v = g_i$ for some $1 \leq i \leq k$, and \emptyset otherwise.

Fix a non-indexed temporal logic \mathbf{TL} , such as $\mathbf{CTL}^* \setminus \mathbf{X}$. We now define what it means for \mathbf{TL} to have the reduction property and the finiteness property. Informally, the reduction property says that if G and G' have the same connectivity (with respect to \mathbf{TL} and only viewing k -tuples \bar{g}, \bar{g}') then P^G and $P^{G'}$ are indistinguishable (with respect to \mathbf{TL} -formulas over process indices in \bar{g}, \bar{g}').

REDUCTION property for \mathbf{TL}

For every positive integer k , process $P \in \mathcal{P}_u$, topologies G, G' , k -tuples \bar{g}, \bar{g}' , if $G|\bar{g} \equiv_{\mathbf{TL}} G'|\bar{g}'$ then $P^G|\bar{g} \equiv_{\mathbf{TL}} P^{G'}|\bar{g}'$.

FINITENESS property for \mathbf{TL}

For every positive integer k , there are finitely many equivalence classes $[G|\bar{g}] \equiv_{\mathbf{TL}}$ where G is an arbitrary topology, and \bar{g} is a k -tuple over V_G .

Theorem 5 (REDUCTION & FINITENESS \implies Cutoffs exist for $\{\forall, \exists\}^k\text{-}\mathbf{TL}$).
If \mathbf{TL} satisfies the REDUCTION property and \mathbf{G} satisfies the FINITENESS property with respect to \mathbf{TL} then for every k , $\mathbf{PMCP}_{\mathbf{G}}(\mathcal{P}_u, \{\forall, \exists\}^k\text{-}\mathbf{TL})$ has a cutoff.

Proof. Fix quantifier prefix $Q_1x_1 \dots Q_kx_k$. We prove that there exist finitely many topologies $G_1, \dots, G_N \in \mathbf{G}$ such that for every $G \in \mathbf{G}$ there is an $i \leq N$ such that for all $P \in \mathcal{P}_u$, and all \mathbf{TL} -formulas φ over atoms $\mathbf{AP}_{\mathbf{pr}} \times \{x_1, \dots, x_k\}$

$$P^G \models Q_1x_1 \dots Q_kx_k. \varphi \iff P^{G_i} \models Q_1x_1 \dots Q_kx_k. \varphi$$

In particular, $c := \max\{|V_{G_i}| : 1 \leq i \leq N\}$ is a cutoff for $\mathbf{PMCP}_{\mathbf{G}}(\mathcal{P}_u, \{\forall, \exists\}^k\text{-}\mathbf{TL})$.

Suppose for simplicity of exposition that $Q_i x_i$ is a quantifier that also expresses that the value of x_i is different from the values of $x_j \in \{x_1, \dots, x_{i-1}\}$.⁹ Fix representatives of $[G|\bar{g}] \equiv_{\mathbf{TL}}$ and let r map $G|\bar{g}$ to the representative of $[G|\bar{g}] \equiv_{\mathbf{TL}}$. Define a function rep that maps $P^G|\bar{g}$ to $P^H|\bar{h}$, where $r(G|\bar{g}) = H|\bar{h}$.

For every $\equiv_{\mathbf{TL}}$ -representative $H|\bar{h}$ (i.e., $H|\bar{h} = r(G|\bar{g})$ for some topology G and k -tuple \bar{g}), introduce a new Boolean proposition $q_{H|\bar{h}}$. By the FINITENESS property of \mathbf{TL} there are finitely many such Boolean propositions, say n .

Define a valuation e_φ (that depends on φ) of these new atoms by

$$e_\varphi(q_{H|\bar{h}}) := \begin{cases} \top & \text{if } P^H|\bar{h} \models \varphi[p_{x_j} \mapsto p @ j] \\ \perp & \text{otherwise.} \end{cases}$$

⁹ All the types of quantifiers defined in Section 2.3, such as $\exists x \in E(y)$, can be dealt with similarly at the cost of notational overhead.

For every $G \in \mathbf{G}$ define Boolean formula $B_G := (\overline{Q_1}g_1 \in V_G) \dots (\overline{Q_k}g_k \in V_G) q_{r(G|\bar{g})}$, where \overline{Q} is the Boolean operation corresponding to Q , e.g., $\exists g_i \in V_G$ is interpreted as $\bigvee_{g \in V_G \setminus \{1, \dots, i-1\}}$.¹⁰

Then (for all P, G and φ)

$$\begin{aligned} P^G &\models Q_1x_1 \dots Q_kx_k. \varphi \\ &\iff Q_1g_1 \in V_G \dots Q_kg_k \in V_G : P^G \models \varphi[p_{x_j} \mapsto p_{g_j}] \\ &\iff Q_1g_1 \in V_G \dots Q_kg_k \in V_G : P^G|\bar{g} \models \varphi[p_{x_j} \mapsto p@j] \\ &\iff Q_1g_1 \in V_G \dots Q_kg_k \in V_G : \text{rep}(P^G|\bar{g}) \models \varphi[p_{x_j} \mapsto p@j] \\ &\iff e_\varphi(B_G) = \top \end{aligned}$$

Here $\varphi[p_{x_j} \mapsto p_{g_j}]$ is the formula resulting from replacing every atom in φ of the form p_{x_j} by the atom p_{g_j} , for $p \in AP_{pr}$ and $1 \leq j \leq k$. Similarly $\varphi[p_{x_j} \mapsto p@j]$ is defined as the formula resulting from replacing (for all $p \in AP_{pr}, j \leq k$) every atom in φ of the form p_{x_j} by the atom $p@j$. The first equivalence is by the definition of semantics of indexed temporal logic; the second is by the definition of $P^G|\bar{g}$; the third is by the REDUCTION property of TL ; the fourth is by the definition of e_φ and rep .

Fix B_{G_1}, \dots, B_{G_N} (with $G_i \in \mathbf{G}$) such that every B_G ($G \in \mathbf{G}$) is logically equivalent to some B_{G_i} . Such a finite set of formulas exists since there are 2^{2^n} Boolean formulas (up to logical equivalence) over n Boolean propositions, and thus at most 2^{2^n} amongst the B_G for $G \in \mathbf{G}$.

By the equivalences above conclude that for every $G \in \mathbf{G}$ there exists $i \leq N$ such that $P^G \models Q_1x_1 \dots Q_kx_k. \varphi$ if and only if $P^{G_i} \models Q_1x_1 \dots Q_kx_k. \varphi$. Thus ' $\forall G \in \mathbf{G}, P^G \models \varphi$ ' is equivalent to ' $\bigwedge_{i \leq N} e_\varphi(B_{G_i})$ ' and so the integer $c := \max\{|V_{G_i}| : 1 \leq i \leq N\}$ is a cutoff for $\text{PMCP}_{\mathbf{G}}(\mathcal{P}_u, \{\forall, \exists\}^k\text{-TL})$. \square

Remark 6. Fix parameterized topology \mathbf{G} , and assume that one can decide if $P^G \models \varphi$ given $\varphi \in \mathcal{F}, G \in \mathbf{G}, P \in \mathcal{P}$. Then the theorem implies that for every $k \in \mathbb{N}$, $\text{PMCP}_{\mathbf{G}}(\mathcal{P}, \{\forall, \exists\}^k\text{-TL})$ is decidable. Further, suppose that given k one could compute the finite set G_1, \dots, G_N . Then by the last sentence in the proof one can compute the cutoff c . In this case, $\text{PMCP}_{\mathbf{G}}(\mathcal{P}, \{\forall, \exists\}^*\text{-TL})$ is decidable.

4 Existence of Cutoffs for k -indexed $\text{CTL}_d^* \setminus X$

The following theorem answers Question 1 from the introduction.

Theorem 7. Let \mathbf{G} be a parameterized topology. Then for all $k, d \in \mathbb{N}$, the problem $\text{PMCP}_{\mathbf{G}}(\mathcal{P}_u, \{\forall, \exists\}^k\text{-CTL}_d^* \setminus X)$ has a cutoff.

Corollary 8. Let \mathbf{G} be a parameterized topology. Then for all $k, d \in \mathbb{N}$, the problem $\text{PMCP}_{\mathbf{G}}(\mathcal{P}_u, \{\forall, \exists\}^k\text{-CTL}_d^* \setminus X)$ is decidable.

¹⁰ Note that in the Boolean propositions G is fixed while $\bar{g} = (g_1, \dots, g_n)$ ranges over $(V_G)^k$ and is determined by the valuation of the variables g_i .

To prove the Theorem it is enough, by Theorem 5, to show that $\{\forall, \exists\}^k\text{-}\text{CTL}_d^*\setminus X$ has the REDUCTION property, and that an arbitrary \mathbf{G} has the FINITENESS property with respect to $\{\forall, \exists\}^k\text{-}\text{CTL}_d^*\setminus X$.

Reduction Theorem. Considering previous results, the following theorem identifies the key reason that a formula holds in a system on a big ring iff it holds in a system on a small ring (cf. [7]), and that a formula holds on a system on an arbitrary topology G iff that formula holds in a system on the network topology of G (in the notation of [4]).

Theorem 9 (Reduction). *For all $d, k \in \mathbb{N}$, topologies G, G' , processes $P \in \mathcal{P}_u$, k -tuples \bar{g} over V_G and k -tuples \bar{g}' over $V_{G'}$:*

$$\text{If } G|\bar{g} \equiv_{\text{CTL}_d^*\setminus X} G'|\bar{g}' \text{ then } P^G|\bar{g} \equiv_{\text{CTL}_d^*\setminus X} P^{G'}|\bar{g}'.$$

The idea behind the proof is to show that paths in P^G can be simulated by paths in $P^{G'}$ (and vice versa). Given a path π in P^G , first project it onto G to get a path ρ that records the movement of the token, then take an equivalent path ρ' in G' which exists since $G|\bar{g} \equiv_{\text{CTL}_d^*\setminus X} G'|\bar{g}'$, and then lift ρ' up to get a path π' in $P^{G'}$ that is equivalent to π . This lifting step uses the assumption that process P is in \mathcal{P}_u , i.e., P cannot control where it sends the token, or from where it receives it. The proof can be found in the full version of the paper [1].

Remark 10. As immediate corollaries we get that the REDUCTION property holds with $\text{TL} = \text{LTL}\setminus X$ (take $d = 1$), $\text{CTL}^*\setminus X$ (since if the assumption holds with $\text{TL} = \text{CTL}^*\setminus X$ then the conclusion holds with $\text{TL} = \text{CTL}_d^*\setminus X$ for all $d \in \mathbb{N}$, and thus also for $\text{TL} = \text{CTL}^*\setminus X$) and, if P is finite, also for $\text{TL} = \text{CTL}\setminus X$ (since $\text{CTL}\setminus X$ and $\text{CTL}^*\setminus X$ agree on finite structures).

Finiteness Theorem. Theorem 4 ([4, Corollary 3]) states that there exists \mathbf{G} such that the problem $\text{PMCP}_{\mathbf{G}}(\mathcal{P}_u, \exists\exists\text{-}\text{CTL}^*\setminus X)$ does not have a cutoff. We observed that the formulas from their result have unbounded nesting depth of path quantifiers. This leads to the idea of stratifying $\text{CTL}^*\setminus X$ by nesting depth.

Recall from Section 2.3 that i) $\text{CTL}_d^*\setminus X$ denotes the syntactic fragment of $\text{CTL}^*\setminus X$ in which formulas have path-quantifier nesting depth at most d ; ii) $M \equiv_{\text{CTL}_d^*\setminus X} N$ iff M and N agree on all $\text{CTL}_d^*\setminus X$ formulas. Write $[M]_{\text{CTL}_d^*\setminus X}$ for the set of all LTSs N such that $M \equiv_{\text{CTL}_d^*\setminus X} N$.

Following the method of Section 3 we prove that the following FINITENESS property holds (where k represents the number of process-index quantifiers in the prenex indexed temporal logic formula).

Remark 11. For ease of exposition we sketch a proof under the assumption that path quantifiers in formulas ignore runs in which the token does not visit every process infinitely often. This is an explicit restriction in [4] and implicit in [7]. In the full version [1] we remove this restriction. For the purpose of this paper this restriction only affects the explicit cutoffs in Theorem 15.

Theorem 12 (Finiteness). *For all positive integers k and d , there are finitely many equivalence classes $[G|\bar{g}]_{\equiv_{\text{CTL}_d^*\setminus X}}$ where G is an arbitrary topology, and \bar{g} is a k -tuple over V_G .*

Proof Idea. We provide an algorithm that given positive integers k, d , topology G , k -tuple \bar{g} over V_G , returns a LTS $\text{CON}_d G | \bar{g}$ such that $G | \bar{g} \equiv_{\text{CTL}_d^* \setminus X} \text{CON}_d G | \bar{g}$. Moreover, we prove that for fixed k, d the range of CON_{d-1} is finite.

Recursively define a marking function μ_d that associates with each $v \in V_G$ a finite set (of finite strings over alphabet $\mu_{d-1}(V_G)$). For the base case define $\mu_0(v) := \Lambda(v)$, the labeling of $G | \bar{g}$. The marking $\mu_d(v)$ stores (representatives) of all strings of μ_{d-1} -labels of paths that start in v and reach some element in \bar{g} . The idea is that $\mu_d(v)$ determines the set of $\text{CTL}_d^* \setminus X$ formulas that hold in $G | \bar{g}$ with initial vertex v , as follows: stitch together these strings, using elements of \bar{g} as stitching points, to get the $\text{CTL}_d^* \setminus X$ types of the infinite paths starting in v . This allows us to define a topology, called the d -contraction $\text{CON}_d G | \bar{g}$, whose vertices are the μ_d -markings of vertices in G . In the full version [1] we prove that $G | \bar{g}$ is $\text{CTL}_d^* \setminus X$ -equivalent to its d -contraction, and that the number of different d -contractions is finite, and depends on k and d .

Definition of d -contraction $\text{CON}_d G | \bar{g}$. Next we define d -contractions.

Marking μ_d . Fix $k, d \in \mathbb{N}$, topology G , and k -tuple \bar{g} over V_G . Let Λ be the labeling-function of $G | \bar{g}$, i.e., $\Lambda(v) = \{i\}$ if $v = g_i$, and $\Lambda(v) = \emptyset$ for $v \notin \bar{g}$. For every vertex $v \in V_G$ define a set $X(v)$ of paths of G as follows: a path $\pi = \pi_1 \dots \pi_t$, say of length t , is in $X(v)$ if π starts in v , none of π_1, \dots, π_{t-1} is in \bar{g} , and $\pi_t \in \bar{g}$. Note that $X(g_i) = \{g_i\}$.

Define the marking μ_d inductively:

$$\mu_d(v) := \begin{cases} \Lambda(v) & \text{if } d = 0 \\ \{\text{destutter}(\mu_{d-1}(\pi_1) \dots \mu_{d-1}(\pi_t)) : \pi_1 \dots \pi_t \in X(v), t \in \mathbb{N}\} & \text{if } d > 0, \end{cases}$$

where $\text{destutter}(w)$ is the maximal substring s of w such that for every two consecutive letters s_i and s_{i+1} we have that $s_i \neq s_{i+1}$. Informally, remove identical consecutive letters of w to get the ‘destuttering’ $\text{destutter}(w)$.

Note that the elements of $\mu_d(v)$ ($d > 0$) are finite strings over the alphabet $\mu_{d-1}(V_G)$. For instance, strings in $\mu_1(v)$ are over the alphabet $\{\{1\}, \{2\}, \dots, \{k\}, \emptyset\}$.

Equivalence relation \sim_d . Vertices $v, u \in V_G$ are d -equivalent, written $u \sim_d v$, if $\mu_d(v) = \mu_d(u)$. We say that \sim_d refines \sim_j if $u \sim_d v$ implies $u \sim_j v$.

Lemma 13. *If $0 \leq j < d$, then \sim_d refines \sim_j .*

Indeed, observe that for all nodes v , all strings in $\mu_d(v)$ start with the letter $\mu_{d-1}(v)$. Thus $\mu_d(v) = \mu_d(u)$ implies that $\mu_{d-1}(v) = \mu_{d-1}(u)$. In other words, if $u \sim_d v$ then $u \sim_{d-1} v$, and thus also $u \sim_j v$ for $0 \leq j < d$.

d-contraction $\text{CON}_d G | \bar{g}$. Define an LTS $\text{CON}_d G | \bar{g}$ called the d -contraction of $G | \bar{g}$ as follows. The nodes of the contraction are the \sim_d -equivalence classes. Put an edge between $[u]_{\sim_d}$ and $[v]_{\sim_d}$ if there exists $u' \in [u]_{\sim_d}, v' \in [v]_{\sim_d}$ and an edge in G from u' to v' . The initial state is $[x]_{\sim_d}$ where x is the initial vertex of G . The label of $[u]_{\sim_d}$ is defined to be $\Lambda(u)$ — this is well-defined because, by Lemma 13, \sim_d refines \sim_0 .

In the full version [1] we prove that $G|\bar{g}$ is $\text{CTL}_d^*\backslash X$ -equivalent to its d -contraction, and that the number of different d -contractions is finite, and depends on k and d . \square

5 Cutoffs for k -index $\text{CTL}^*\backslash X$ and Concrete Topologies

The following two Theorems answer Question 2 from the introduction regarding PMCP for specifications from $\{\forall, \exists\}^*\text{-}\text{CTL}^*\backslash X$.

First, PMCP is undecidable for certain (pathological) parameterized topologies \mathbf{G} and specifications from $\{\forall, \exists\}^*\text{-}\text{CTL}^*\backslash X$.

Theorem 14. *There exists a process $P \in \mathcal{P}_u$, and parameterized topologies \mathbf{G}, \mathbf{H} , such that the following PMCP problems are undecidable*

1. $\text{PMCP}_{\mathbf{G}}(\{P\}, \{\forall, \exists\}^*\text{-}\text{LTL}\backslash X)$.
2. $\text{PMCP}_{\mathbf{H}}(\{P\}, \{\forall, \exists\}^2\text{-}\text{CTL}\backslash X)$.

Moreover, \mathbf{G} and \mathbf{H} can be chosen to be computable sets of topologies.

Second, PMCP is decidable for certain (regular) parameterized topologies and specifications from $\{\forall\}^*\text{-}\text{CTL}^*\backslash X$. This generalises results from Emerson and Namjoshi [7] who show this result for $k = 1, 2$ and uni-directional ring topologies. By Remark 11 these cutoffs apply under the assumption that we ignore runs that do not visit every process infinitely often.

Theorem 15. *If \mathbf{G} is as stated, then $\text{PMCP}_{\mathbf{G}}(\mathcal{P}_u, \{\forall\}^k\text{-}\text{CTL}^*\backslash X)$ has the stated cutoff.*

1. *If \mathbf{G} is the set of uni-directional rings, then $2k$ is a cutoff.*
2. *If \mathbf{G} is the set of bi-directional rings, then $2k$ is a cutoff.*
3. *If \mathbf{G} is the set of cliques, then $k + 1$ is a cutoff.*
4. *If \mathbf{G} is the set of stars, then $k + 1$ is a cutoff.*

Consequently, for each \mathbf{G} listed, $\text{PMCP}_{\mathbf{G}}(\mathcal{P}_u, \{\forall\}^*\text{-}\text{CTL}^*\backslash X)$ is decidable.

This theorem is proved following Remark 6: given k, d , we compute a set $G_1, \dots, G_N \in \mathbf{G}$ such that every B_G for $G \in \mathbf{G}$ is logically equivalent to some B_{G_i} , where the Boolean formula B_G is defined as $\bigwedge_{\bar{g}} q_{\text{CON}_d G|\bar{g}}$. To do this note that B_G is logically equivalent to B_H if and only if $\{\text{CON}_d G|\bar{g} : \bar{g} \in V_G\} = \{\text{CON}_d H|\bar{h} : \bar{h} \in V_H\}$ (this is where we use that there is no quantifier alternation). So it is sufficient to prove that, if c is the stated cutoff,

$$|V_G|, |V_H| \geq c \implies \{\text{CON}_d G|\bar{g} : \bar{g} \in V_G\} = \{\text{CON}_d H|\bar{h} : \bar{h} \in V_H\}$$

To illustrate how to do this we analyse the case of uni-directional rings and cliques (the other cases are similar).

Uni-directional rings. Suppose \mathbf{G} are the uni-directional rings and let $G \in \mathbf{G}$. Fix a k -tuple of distinct elements of V_G , say (g_1, g_2, \dots, g_k) . Define a function $f : V_G \rightarrow \{g_1, \dots, g_k\}$ that maps v to the first element of \bar{g} on the path $v, v + 1, v + 2, \dots$ (addition is mod $|V_G|$). In particular $f(g_i) = g_i$ for $i \in [k]$. In

the terminology of the proof of Theorem 12, $X(v)$ consists of the simple path $v, v+1, \dots, f(v)$.

We now describe μ_d . Clearly $\mu_d(g_i) = \{\mu_{d-1}(g_i)\}$. By induction on d one can prove that if $v \notin \bar{g}$ with $f(v) = g_j$ then $\mu_d(v) = \{\mu_{d-1}(v) \cdot \mu_{d-1}(g_j)\}$. So for every $d > 1$, the equivalences \sim_d and \sim_1 coincide.

d	0	1	2	\dots
$\mu_d(v)$ for $v = g_i$	$\{\{i\}\}$	$\{\{i\}\}$	$\{\{\{i\}\}\}$	\dots
$\mu_d(v)$ if $v \notin \bar{g}$ and $f(v) = g_j$	\emptyset	$\{\emptyset \cdot \{j\}\}$	$\{\{\emptyset \cdot \{j\}\} \cdot \{\{j\}\}\}$	\dots

Thus for every $k \in \mathbb{N}$, the d -contraction $\text{CON}_d G | \bar{g}$ is a ring of size at most $2k$ (in particular, it is independent of d). In words, the d -contraction of G is the ring resulting by identifying adjacent elements not in \bar{g} . It is not hard to see that if G, H are rings such that $|V_G|, |V_H| \geq 2k$ then for every \bar{g} there exists \bar{h} such that $\text{CON}_d G | \bar{g} = \text{CON}_d H | \bar{h}$.

Cliques. Fix $n \in \mathbb{N}$. Let G be a clique of size n . That is: $V_G = [n]$ and $(i, j) \in E_G$ for $1 \leq i \neq j \leq n$. Fix a k -tuple of distinct elements of V_G , say (g_1, g_2, \dots, g_k) . We now describe $\mu_d(v)$. Clearly $\mu_d(g_i) = \{\mu_{d-1}(g_i)\}$ and for $v \notin \bar{g}$ we have $\mu_d(v) = \{\mu_{d-1}(v) \cdot \mu_{d-1}(j) : j \in [k]\}$.

It is not hard to prove that for every $k \in \mathbb{N}$, the d -contraction $\text{CON}_d G | \bar{g}$ is the clique of size $k+1$. In words, the d -contraction of G results from G by identifying all vertices not in \bar{g} . It is not hard to see that if G, H are cliques such that $|V_G|, |V_H| \geq k+1$ then for every \bar{g} there exists \bar{h} such that $\text{CON}_d G | \bar{g} = \text{CON}_d H | \bar{h}$.

Remark 16. For cliques and stars $k+1$ is also the cutoff for $\{\forall, \exists\}^k\text{-CTL}^* \setminus X$.

Remark 17. $2k$ is not the cutoff for uni-rings and $\{\forall, \exists\}^k\text{-CTL}^* \setminus X$ as stated in [12, Corollary 2]. The formula $\forall i \forall j \exists k. adj(k, i) \vee adj(k, j)$, where $adj(k, i) := tok_i \rightarrow tok_i \cup tok_k \vee tok_k \rightarrow tok_k \cup tok_i$, holds in ring of size 6 but not in 7.

6 There are No Cutoffs for Direction-Aware Systems

In the following, we consider systems where processes can choose which directions are used to send or receive the token, i.e., process templates are from \mathcal{P}_{snd} , \mathcal{P}_{rcv} , or $\mathcal{P}_{\text{sndrcv}}$. Let \mathbf{B} be the parameterized topology of all bi-directional rings, with directions cw (clockwise) and ccw (counter-clockwise). The following theorem answers Question 3 from the introduction.

Theorem 18. 1. The problem $\text{PMCP}_B(\mathcal{P}_{\text{sndrcv}}, \forall\text{-LTL} \setminus X)$ is undecidable.
2. For \mathcal{F} equal to $\{\forall\}^9\text{-LTL} \setminus X$ or $\{\exists\}^9\text{-CTL} \setminus X$, and $\mathcal{P} \in \{\mathcal{P}_{\text{snd}}, \mathcal{P}_{\text{rcv}}\}$, there exists a parameterized topology \mathbf{G} such that $\text{PMCP}_G(\mathcal{P}, \mathcal{F})$ is undecidable.

Proof Idea. In all cases we reduce the non-halting problem of two-counter machines (2CMs) to the PMCP. The idea is that one process, the *controller*, simulates the finite-state control of the 2CM. The other processes, arranged in a chain or a ring, are *memory processes*, collectively storing the counter values with a

fixed memory per process. This allows a given system to simulate a 2CM with bounded counters. Since a 2CM terminates if and only if it terminates for some bound on the counter values, we can reduce the non-halting problem of 2CMs to the PMCP. The main work is to show that the controller can issue commands, such as ‘increment counter 1’ and ‘test counter 1 for zero’. We give a detailed proof sketch for part 1 of the theorem, and then outline a proof for part 2.

1. \forall - $LTL \setminus X$ and \mathcal{P}_{sndrcv} in bi-directional rings.

The process starting with the token becomes the controller, all others are memory, each storing one bit for each counter of the 2CM. The current value of a counter c is the total number of corresponding bits (c -bits) set to 1. Thus, a system with n processes can store counter values up to $n - 1$.

Fix a numbering of 2CM-commands, say $0 \mapsto$ ‘increment counter 1’, $1 \mapsto$ ‘decrement counter 1’, $2 \mapsto$ ‘test counter 1 for zero’, etc. Every process has a command variable that represents the command to be executed when it receives the token from direction ccw .

If the controller sends the token in direction cw , the memory processes will increment (mod 6) the command variable, allowing the controller to encode which command should be executed. Every process just continues to pass the token in direction cw , until it reaches the controller again.

If the controller sends the token in direction ccw , then the memory processes try to execute the command currently stored. If it is an ‘increment counter c ’ or ‘decrement counter c ’ command, the memory process tries to execute it (by incrementing/decrementing its c -bit). If the process cannot execute the command (because the c -bit is already 1 for an increment, or 0 for a decrement), then it passes the token along direction ccw and remembers that a command is being tried. If the token reaches a memory process which can execute the command, then it does so and passes the token back in direction cw . The processes that remembered that a command is being tried will receive the token from direction cw , and know that the command has been successfully executed, and so will the controller. If the controller gets the token from ccw , the command failed. In this case, the controller enters a loop in which it just passes the token in direction cw (and no more commands are executed).

If the command stored in the memory processes is a ‘test for zero counter c ’, then the processes check if their c -bit is 0. If this is the case, it (remembers that a command is being tried and) passes the token to the next process in direction ccw . If the token reaches a process for which the c -bit is 1, then this process sends the token back in direction cw . Other memory processes receiving it from cw (and remembering that the command is being tried), pass it on in direction cw . In this case, the controller will receive the token from cw and know that counter c is not zero. On the other hand, if all memory processes store 0 in their c -bit, then they all send the token in direction ccw . Thus, the controller will receive it from ccw and knows that counter c currently is zero. To terminate the command, it sends the token in direction cw , and all processes (which remembered that a command is being tried), know that execution of this command is finished.

With the description above, a system with $n - 1$ memory processes can simulate a 2CM as long as counter values are less than n . Let HALT be an atomic proposition that holds only in the controller's halting states. Then solving the PMCP for $\forall i \mathbb{G} \neg \text{HALT}_i$ amounts to solving the non-halting problem of the 2CM.
 $\mathcal{Z}, \{\forall\}^9\text{-}\text{LTL} \setminus X$ and \mathcal{P}_{snd} .

We give a proof outline. In this case there are $2n$ memory processes, n for each counter $c \in \{1, 2\}$. The remaining 9 processes are special and called 'controller', 'counter c is zero', 'counter c is not zero', 'counter c was incremented', and 'counter c was decremented'. When the controller wants to increment or decrement counter c , it sends the token nondeterministically to some memory process for counter c . When the controller wants to test counter c for zero, it sends the token to the first memory process. When a memory process receive the token it does not know who sent it, and in particular does not know the intended command. Thus, it non-deterministically takes the appropriate action for one of the possible commands. If its bit is set to 0 then it either i) increments its bit and sends the token to a special process 'counter c was incremented', or ii) it sends the token to the next memory node in the chain, or to the special process 'counter c is zero' if it is the last in the chain. If its bit is set to 1 then it either i) decrements its bit and sends the token to a special process 'counter c was decremented', or ii) sends the token to a special process 'counter c is not zero'.

Even though incoming directions are not available to the processes, we can write the specification such that, out of all the possible non-deterministic runs, we only consider those in which the controller receives the token from the expected special node (the formula requires one quantified index variable for each of the special nodes). So, if the controller wanted to increment counter c it needs to receive the token from process 'counter c was incremented'. If the controller receives the token from a different node, it means that a command was issued but not executed correctly, and we disregard this run. Otherwise, the system of size $2n + 1$ correctly simulates the 2CM until one of the counter values exceeds n . As before, parameterized model checking of this construction would allow us to solve the non-halting problem for 2CMs. \square

7 Extensions

There are a number of extensions of direction-unaware TPSs for which the theorems that state existence of cutoffs (Theorems 7 and 15) still hold. We describe these in order to highlight assumptions that make the proofs work:

1. Processes can be infinite-state.
2. The EN-restriction on the process template P can be relaxed: replace item *vii*) in Definition 2.1 by "For every state q that has the token there is a finite path $q \dots q'$ such that q' does not have the token, and for every q that does not have the token there is a finite path $q \dots q'$ such that q' has the token".
3. One can further allow *direction-sensing* TPSs, which is a direction-aware TPS with an additional restriction on the process template: "If $q \xrightarrow{d} q' \in \delta$ for some direction $d \in \text{Dir}_{\text{snd}}$, then for every $d \in \text{Dir}_{\text{snd}}$ there exists a transition

$q \xrightarrow{d} q'' \in \delta''$; and a similar statement for Dir_{rcv} . Informally: we can allow processes to change state according to the direction that the token is (non-deterministically) sent to or received, but the processes are not allowed to block any particular direction.

4. One can further allow the token to carry a value but with the strong restriction that from every state that has the token and every value v there is a path of internal actions in P which eventually sends the token with value v , and the same for receiving.

These conditions on P all have the same flavor: they ensure that a process can not choose what information to send/receive, whether that information is a value on the token or a direction for the token.

8 Related work

This paper is based on [14,7,4] (as described in the introduction). However, there are several other relevant papers.

Emerson and Kahlon [6] consider token-passing in uni- and bi-directional rings, where processes are direction-aware and tokens carry messages (but can only be changed a bounded number of times). However, the provided cutoff theorems only hold for specifications that talk about two processes (in a unidirectional ring) or one process (in a bi-directional ring), and process implementations need to be deterministic. Furthermore, cutoffs depend on the complexity of the process implementation.

German and Sistla [11] provide cutoffs for the PMCP for systems with pairwise synchronization. Although pairwise synchronization can simulate token-passing, their cutoff results are restricted to cliques and 1-indexed LTL. Moreover, their proof uses vector-addition systems with states and their cutoff depends on the process template and the specification formula. German and Sistla [11, Section 6] also consider non-prenex formulas with the restriction that exactly one index variable name occurs, e.g., $\exists i.\text{Req}_i \rightarrow \mathsf{F} \exists i.\text{Grant}_i$, and prove that the resulting PMCP is undecidable.

Delzanno et al. [5] study a model of broadcast protocols on arbitrary topologies, in which a process can synchronize with all of its available neighbours ‘at once’ by broadcasting a message (from a finite set of messages). They prove undecidability of PMCP for systems with arbitrary topologies and 1-indexed safety properties, and that the problem becomes decidable if one restricts the topologies to ‘graphs with bounded paths’ (such as stars). Their proof uses the machinery of well-structured transitions systems, and no cutoffs are provided. They also show undecidability of the PMCP in the case of non-prenex indexed properties of the form $\mathsf{G}(\exists i.s(i) \in B)$ on general and the restricted topologies.

9 Summary

The goal of this work was to find out under what conditions there are cutoffs for temporal logics and token-passing systems on general topologies. We found that

stratifying prenex indexed $\text{CTL}^*\backslash X$ by nesting-depth of path quantifiers allowed us to recover the existence of cutoffs; but that there are no cutoffs if the processes are allowed to choose the direction of the token. In all the considered cases where there is no cutoff we show that the PMCP problem is actually undecidable.

Our positive results are provided by a construction that generalizes and unifies the known positive results, and clearly decomposes the problem into two aspects: tracking the movement of the token through the underlying topology, and simulating the internal states of the processes that the specification formula can see. The construction yields small cutoffs for common topologies (such as rings, stars, and cliques) and specifications from prenex indexed $\text{CTL}^*\backslash X$.

Acknowledgments. We thank Roderick Bloem for detailed comments on numerous drafts and Krishnendu Chatterjee for important comments regarding the structure of the paper. We thank Roderick Bloem, Igor Konnov, Helmut Veith, and Josef Widder for discussions at an early stage of this work that, in particular, pointed out the relevance of direction-unawareness in [4].

References

1. Aminof, B., Jacobs, S., Khalimov, A., Rubin, S.: Parameterized Model Checking of Token-Passing Systems (2013), pre-print on arxiv.org
2. Baier, C., Katoen, J.P., et al.: Principles of model checking, vol. 26202649. MIT press Cambridge (2008)
3. Browne, M.C., Clarke, E.M., Grumberg, O.: Reasoning about networks with many identical finite state processes. *Inf. Comput.* 81, 13–31 (April 1989)
4. Clarke, E., Talupur, M., Touili, T., Veith, H.: Verification by network decomposition. In: CONCUR 2004. vol. 3170, pp. 276–291 (2004)
5. Delzanno, G., Sangnier, A., Zavattaro, G.: Parameterized verification of ad hoc networks. In: CONCUR. LNCS, vol. 6269, pp. 313–327 (2010)
6. Emerson, E.A., Kahlon, V.: Parameterized model checking of ring-based message passing systems. In: CSL. LNCS, vol. 3210, pp. 325–339. Springer (2004)
7. Emerson, E.A., Namjoshi, K.S.: On reasoning about rings. *Int. J. Found. Comput. Sci.* 14(4), 527–550 (2003)
8. Emerson, E.A., Sistla, A.P.: Symmetry and model checking. In: CAV. pp. 463–478 (1993)
9. Emerson, E., Namjoshi, K.: Reasoning about rings. In: POPL. pp. 85–94 (1995)
10. Esparza, J., Finkel, A., Mayr, R.: On the verification of broadcast protocols. *Logic in Computer Science, Symposium on* 0, 352 (1999)
11. German, S.M., Sistla, A.P.: Reasoning about systems with many processes. *J. ACM* 39(3), 675–735 (1992)
12. Khalimov, A., Jacobs, S., Bloem, R.: Towards efficient parameterized synthesis. In: Giacobazzi, R., Berdine, J., Mastroeni, I. (eds.) VMCAI, LNCS, vol. 7737, pp. 108–127. Springer Berlin Heidelberg (2013)
13. Shamir, S., Kupferman, O., Shamir, E.: Branching-depth hierarchies. ENTCS 39(1), 65 – 78 (2003)
14. Suzuki, I.: Proving properties of a ring of finite-state machines. *Inf. Process. Lett.* 28(4), 213–214 (Jul 1988)

Parameterized Model Checking of Rendezvous Systems ^{*}

Benjamin Aminof¹, Tomer Kotek², Sasha Rubin², Francesco Spegni³, Helmut Veith²

¹ IST Austria

² TU Wien, Austria

³ UnivPM Ancona, Italy

Abstract. A standard technique for solving the parameterized model checking problem is to reduce it to the classic model checking problem of finitely many finite-state systems. This work considers some of the theoretical power and limitations of this technique. We focus on concurrent systems in which processes communicate via pairwise rendezvous, as well as the special cases of disjunctive guards and token passing; specifications are expressed in indexed temporal logic without the next operator; and the underlying network topologies are generated by suitable Monadic Second Order Logic formulas and graph operations. First, we settle the exact computational complexity of the parameterized model checking problem for some of our concurrent systems, and establish new decidability results for others. Second, we consider the cases that model checking the parameterized system can be reduced to model checking some fixed number of processes, the number is known as a cutoff. We provide many cases for when such cutoffs can be computed, establish lower bounds on the size of such cutoffs, and identify cases where no cutoff exists. Third, we consider cases for which the parameterized system is equivalent to a single finite-state system (more precisely a Büchi word automaton), and establish tight bounds on the sizes of such automata.

1 Introduction

Many concurrent systems consist of an arbitrary number of identical processes running in parallel, possibly in the presence of an environment or control process. The parameterized model checking problem (PMCP) for concurrent systems is to decide if a given temporal logic specification holds irrespective of the number of participating processes.

Although the PMCP is undecidable in general (see [1,2]) for some combinations of communication primitives, network topologies, and specification languages, it is often proved decidable by a reduction to model checking finitely

* The second, third, fourth and fifth authors were supported by the Austrian National Research Network S11403-N23 (RiSE) of the Austrian Science Fund (FWF) and by the Vienna Science and Technology Fund (WWTF) through grants PROSEED, ICT12-059, and VRG11-005.

many finite-state systems [3,4,2,5,6]. In many of these cases it is even possible to reduce the problem of whether a parameterized system satisfies a temporal specification for any number of processes to the same problem for systems with at most c processes. The number c is known as a *cutoff* for the parameterized system. In other cases the reduction produces a single finite-state system, often in the form of an automaton such as a Büchi automaton, that represents the set of all execution traces of systems of all sizes.

The goal of this paper is to better understand the power and limitations of these techniques, and this is done by addressing three concrete questions.

Question 1: For which combinations of communication primitive, specification language, and network topologies is the PMCP decidable? In the decidable cases, what is the computational complexity of the PMCP?

In case a cutoff c exists, the PMCP is decidable by a reduction to model checking c many finite-state systems. The complexity of this procedure depends on the size of the cutoff. Thus we ask:

Question 2: When do cutoffs exist? In case a cutoff exists, can it be computed? And if so, what is a lower bound on the cutoff?

The set of execution traces of a parameterized system (for a given process type P) is defined as the projection onto the local states of P of all (infinite) runs of systems of all sizes.⁴ In case this set is ω -regular, one can reduce the PMCP of certain specifications (including classic ones such as coverability) to the language containment problem for automata (this is the approach taken in [3, Section 4]). Thus we ask:

Question 3: Is the set of executions of the system ω -regular? If so, what is a lower bound on the sizes of the non-deterministic Büchi word automata recognizing the set of executions?

System model. In order to model and verify a concurrent system we should specify three items: (i) the communication primitive, (ii) the specification language, and (iii) the set of topologies.

We focus on concurrent systems in which processes communicate via pairwise rendezvous [3], as well as two other communication primitives (expressible in terms of pairwise rendezvous), namely disjunctive guards [4] and token-passing systems [2,5,6]. Two special cases are systems with one process template U (in other words, all processes run the same code), and systems with two process templates C, U in which there is exactly one copy of C ; in other words, all processes run the same code, except for one (which is called the controller).

Specifications of parameterised systems are typically expressed in indexed temporal logic [7] which allows one to quantify over processes (e.g., $\forall i \neq j. \text{AG}(\neg(\text{critical}, i) \vee \neg(\text{critical}, j))$) says that no two processes are in their critical sections at the same time). We focus on a fragment of this logic where the process quantifiers only appear at the front of a temporal logic formula — al-

⁴ Actually we consider the destuttering of this set, as explained in Section 2.5.

lowing the process quantifiers to appear in the scope of path quantifiers results in undecidability even with no communication between processes [8].

The sets of topologies we consider all have either bounded tree-width, or more generally bounded clique-width, and are expressible in one of three ways. (1) Using MSO, a powerful and general formalism for describing sets of topologies, which can express e.g. planarity, acyclicity and ℓ -connectivity. (2) As *iteratively constructible* sets of topologies, an intuitive formalism which creates graph sequences by iterating graph operations [9]. Many typical classes of topologies (e.g., all rings, all stars, all cliques) are iteratively constructible. (3) As *clique-like* sets of topologies, which includes the set of cliques and the set of stars, but excludes the set of rings. Iteratively constructible and clique-like sets of topologies are MSO-definable, the former in the presence of certain auxiliary relations.

Prior work and our contributions. For each communication primitive (rendezvous, disjunctive guards, token passing) and each question (decidability and complexity, cutoffs, equivalent automata) we summarise the known answers and our contributions. Obviously, the breadth of questions along these axis is great, and we had to limit our choices as to what to address. Thus, this article is not meant to be a comprehensive taxonomy of PMCP. That is, it is not a mapping of the imaginary hypercube representing all possible choices along these axis. Instead, we started from the points in this hypercube that represent the most prominent known results and, guided by the three main questions mentioned earlier, have explored the unknown areas in each point's neighborhood.

Pairwise Rendezvous.

Decidability and Complexity: The PMCP for systems which communicate by pairwise rendezvous, on clique topologies, with a controller C , for 1-index $LTL \setminus X$ specifications is EXPSPACE-complete [3,10] (and PSPACE without a controller [3, Section 4]). We show the PMCP is undecidable if we allow the more general 1-index $CTL^* \setminus X$ specifications. Thus, for the results on pairwise rendezvous we fix the specification language to be 1-index $LTL \setminus X$. We introduce sets of topologies that naturally generalise cliques and stars, and exclude rings (the PMCP is already undecidable for uni-directional rings and 1-index safety specifications [1,2]), which we call *clique-like* sets of topologies, and show that the PMCP of 1-index $LTL \setminus X$ on clique-like topologies is EXPSPACE-complete (PSPACE-complete without a controller). We also prove that the program complexity is EXPSPACE-complete (respectively PTIME).

Cutoffs: We show that even for clique topologies there are not always cutoffs.

Equivalent automata: We prove that the set of (destuttered) executions of systems with a controller are not, in general, ω -regular, already for clique topologies. On the other hand, we extend the known result that the set of (destuttered) executions for systems with only user processes U (i.e., without a controller) is ω -regular for clique topologies [3] to clique-like topologies, and give an effective construction of the corresponding Büchi automaton.

Disjunctive guards.

In this section we focus on clique topologies and 1-index $\text{LTL} \setminus X$ specifications. Though we sometimes consider more general cases (as in Theorem 10), we postpone these cases for future work.

Decidability and Complexity: We show the PMCP is undecidable if we allow 1-index $\text{CTL}^* \setminus X$ specifications, already for clique topologies. We prove that for systems with a controller the complexity of the PMCP is PSPACE-complete and the program complexity is coNP-complete, whereas for systems without a controller the complexity is PSPACE-complete and the program complexity is in PTIME. We note that the PTIME and PSPACE upper bounds follow from [3,4], although we improve the time complexity for the case with a controller.

Cutoffs: Cutoffs exist for such systems and are of size $|U| + 2$ [4]. We prove these cutoffs are tight.

Equivalent automaton: We prove that the set of (destuttered) executions is accepted by an effectively constructible Büchi automaton of size $O(|C| \times 2^{|U|})$. It is very interesting to note that this size is smaller than the smallest system size one gets (in the worst-case) from the cutoff result, namely $|C| \times |U|^{|U|+2}$. Hence, the PMCP algorithm obtained from the cutoff is less efficient than the one obtained from going directly to a Büchi automaton. As far as we know, this is the first theoretical proof of the existence of this phenomenon. We also prove that, in general, our construction is optimal, i.e., that in some cases every automaton for the set of (destuttered) executions must be of size $2^{\Omega(|U|+|C|)}$.

Token passing systems.

In this section we focus on MSO-definable set of topologies of bounded tree-width or clique-width, as well as on iteratively-constructible sets of topologies.

Decidability and Complexity: We prove that the PMCP is decidable for indexed $\text{CTL}^* \setminus X$ on such topologies. This considerably generalises the results of [6], where decidability for this logic was shown for a few concrete topologies such as rings and cliques.

Cutoffs: For the considered topologies and indexed $\text{CTL}^* \setminus X$ we prove that the PMCPs have *computable* cutoffs. From [6] we know that there is a (computable) set of topologies and a system template such that there is no algorithm that given an indexed $\text{CTL}^* \setminus X$ formula can compute the associated cutoff (even though a cutoff for *the given formula* always exists). This justifies our search of sets of topologies for which the PMCP for $\text{CTL}^* \setminus X$ has computable cutoffs. We also give a lower bound on cutoffs for iteratively-constructible sets and indexed $\text{LTL} \setminus X$.

Equivalent automaton: Our ability to compute cutoffs for 1-index $\text{LTL} \setminus X$ formulas and the considered topologies implies that the (destuttered) sets of execution traces are ω -regular, and the construction of Büchi automata which compute these traces is effective.

Due to space limitations, in many cases proofs/sketches are not given, and only a statement of the basic technique used for the proof is given. The reader is referred to the full version of the article for more details.

2 Definitions and Preliminaries

A *labeled transition system* (*LTS*) is a tuple $(S, R, I, \Phi, AP, \Sigma)$, where S is the set of *states*, $R \subseteq S \times \Sigma \times S$ is the *transition relation*, $I \subseteq S$ are the *initial states*, $\Phi : S \rightarrow 2^{AP}$ is the *state-labeling*, AP is a set of *atomic propositions* or *atoms*, and Σ is the *transition-labels alphabet*. When AP and Σ are clear from the context we drop them. A *finite LTS* is an LTS in which S, R, Σ are finite and $\Phi(s)$ is finite for every $s \in S$. Transitions $(s, a, s') \in R$ may be written $s \xrightarrow{a} s'$. A *transition system* (*TS*) (S, R, I, Σ) is an LTS without the labeling function and without the set of atomic propositions. A *run* is an infinite path that starts in an initial state. For a formal definition of path, state-labeled path, action-labeled path, refer to the full version of this paper.

2.1 Process Template, Topology, Pairwise Rendezvous System

We define how to (asynchronously) compose processes that communicate via pairwise rendezvous into a single system. We consider time as being discrete (i.e. not continuous). Processes are not necessarily identical, but we assume only a finite number of different process types. Roughly, at every vertex of a topology (a directed graph with vertices labeled by process types) there is a process of the given type running; at every time step either, and the choice is nondeterministic, exactly one process makes an internal transition, or exactly two processes with an edge between them in the topology instantaneously synchronize on a message (sometimes called an action) $m \in \Sigma_{\text{sync}}$. The sender of the message m performs an $m!$ transition, and the receiver an $m?$ transition. Note that the sender can not direct the message to a specific neighbouring process (nor can the receiver choose from where to receive it), but the pair is chosen non-deterministically.⁵

Fix a countable set of atoms (also called atomic propositions) AP_{pr} . Fix a finite synchronization alphabet Σ_{sync} (that does not include the symbol τ), and define the *communication alphabet* $\Sigma = \{m!, m? \mid m \in \Sigma_{\text{sync}}\}$.

Process Template, System Arity, System Template. A *process template* is a finite LTS $P = (S, R, \{\iota\}, \Phi, AP_{\text{pr}}, \Sigma \cup \{\tau\})$. Since AP_{pr} and the transition-labels alphabet are typically fixed, we will omit them. The *system arity* is a natural number $r \in \mathbb{N}$. It refers to the number of different process types in the system. A *(r-ary) system template* is a tuple of process templates $\bar{P} = (P_1, \dots, P_r)$ where r is the system arity. The process template $P_i = (S_i, R_i, \{\iota_i\}, \Phi_i)$ is called the *ith process template*.

Topology G . An *r-topology* is a finite structure $G = (V, E, T_1, \dots, T_r)$ where $E \subseteq V \times V$, and the $T_i \subseteq V$ partition V . The *type* of $v \in V$ denoted $\text{type}(v)$ is the unique j such that $v \in T_j$. We might write V_G, E_G and type_G to stress G .

We sometimes assume that $V := \{1, \dots, n\}$ for some $n \in \mathbb{N}$. For instance, an *r-ary clique topology* with $V = \{1, \dots, n\}$ has $E = \{(i, j) \in [n]^2 \mid i \neq j\}$ (and

⁵ In models in which we allow processes to send in certain directions, e.g., send left and send right in a bi-directional ring, then PMCP is quickly undecidable [6].

some partition of the nodes into sets T_1, \dots, T_r ; and the *1-ary ring topology* with $V = \{1, \dots, n\}$ has $E = \{(i, j) \in [n]^2 \mid j = i + 1 \bmod n\}$ and $T_1 = V$.

(Pairwise-Rendezvous) System. Given system arity r , system template $\bar{P} = (P_1, \dots, P_r)$ with $P_i = (S_i, R_i, \{\iota_i\}, \Phi_i)$, and r -topology $G = (V, E, \bar{T})$, define the *system* \bar{P}^G as the LTS $(Q, \Delta, Q_0, \Lambda, \text{AP}_{\text{pr}} \times V, \Sigma_{\text{sync}} \cup \{\tau\})$ where

- The set Q is the set of functions $f : V \rightarrow \cup_{i \leq r} S_i$ such that $f(v) \in S_i$ iff $\text{type}(v) = i$ (for all $v \in V, i \leq r$). Such functions (sometimes written as vectors) are called *configurations*.
- The set Q_0 consists of the unique *initial configuration* f_ι defined as $f_\iota(v) = \iota_{\text{type}(v)}$ (for all $v \in V$).
- The set of *global transitions* Δ are tuples $(f, m, g) \in Q \times (\Sigma_{\text{sync}} \cup \{\tau\}) \times Q$ where one of the following two conditions hold:
 - $m = \tau$ and there exists $v \in V$ such that $f(v) \xrightarrow{\tau} g(v)$ is a transition of the process template $P_{\text{type}(v)}$, and for all $w \neq v$, $f(w) = g(w)$; this is called an *internal transition*,
 - $m \in \Sigma_{\text{sync}}$ and there exists $v \neq w \in V$ with $(v, w) \in E$ such that $f(v) \xrightarrow{m!} g(v)$ and $f(w) \xrightarrow{m?} g(w)$ and for all $z \notin \{v, w\}$, $f(z) = g(z)$; this is called a *synchronous transition*. We say that the process at v *sends the message* m and the process at w *receives the message* m .
- The labeling function $\Lambda : Q \rightarrow 2^{\text{AP}_{\text{pr}} \times V}$ is defined by $(p, v) \in \Lambda(f) \iff p \in \Phi_{\text{type}(v)}(f(v))$ (for all configurations f , atoms $p \in \text{AP}_{\text{pr}}$ and vertices $v \in V$).

In words then, a topology of size n specifies n -many processes, which processes have the same type, and how the processes are connected. In the internal transition above only the process at vertex v makes a transition, and in the synchronous transition above only the process at vertex v and its neighbour at w make a transition. Let $\pi = f_0 f_1 \dots$ be a state-labeled path in \bar{P}^G . The *projection of π to vertex $v \in V$* , written $\text{proj}_v(\pi)$, is the sequence $f_0(v) f_1(v) \dots$ of states of $P_{\text{type}(v)}$. If $\text{type}(v) = j$ we say that the *vertex v runs (a copy of) the process P_j* , or that *the process at v is P_j* .

2.2 Disjunctively-Guarded System, and Token Passing System

We define guarded protocols and token-passing systems as restricted forms of pairwise rendezvous systems. In fact, the restrictions are on the system template and the synchronization alphabet. Write $P_i = (S_i, R_i, \{\iota_i\}, \Phi_i, \text{AP}_{\text{pr}}, \Sigma \cup \{\tau\})$.

Disjunctively-Guarded System Template. A system \bar{P}^G is *disjunctively-guarded* if \bar{P} is. A system template \bar{P} is *disjunctively-guarded* if **(i)** The state sets of the process templates are pairwise disjoint, i.e., $S_i \cap S_j = \emptyset$ for $1 \leq i < j \leq r$. **(ii)** The transition-labels alphabet Σ is $\{\tau\} \cup \{q!, q? \mid q \in \cup_{i \leq r} S_i\}$ **(iii)** For every state $s \in S$, there is a transition labeled $s \xrightarrow{s?} s$. **(iv)** For every state $s \in S$, the only transitions labeled $s?$ are of the form $s \xrightarrow{s?} s$. Intuitively, in this kind of systems a process can decide to move depending on the local state of some

neighbor process, but it cannot relate the state of any two processes at the same time, nor it can force another process to move from its local state. Our definition of disjunctively-guarded systems on a clique topology is a reformulation of the definition of concrete system in [4, Section 2].

Token Passing System. One can express a token passing system (TPS) as a special case of pairwise rendezvous. In this work we only consider the case of a single valueless token, whose formal definition can be found in [6,2]. A token passing system (TPS) \overline{P}^G can be thought of the asynchronous parallel composition of the processes templates in \overline{P} over topology G according to the types of vertices. At any time during the computation, exactly one vertex has the token. The token starts with the unique process in P_1 . At each time step either exactly one process makes an internal transition, or exactly two processes synchronize when one process sends the token to another along an edge of G .

2.3 Indexed Temporal Logic

We assume the reader is familiar with the syntax and semantics of CTL* and LTL. Indexed temporal logics were introduced by [7] to model specifications of certain distributed systems. They are obtained by adding *vertex quantifiers* to a given temporal logic over indexed atomic propositions. For example, in a system with two process templates, the formula $\forall i : type(i) = 1. \text{AG}((good, i))$ states that every process of type 1 on all computations at all points of time satisfies the atom *good*. In a system with one process template, the formula $\forall i \neq j. \text{AG}(\neg(critical, i) \vee \neg(critical, j))$ states that it is never the case that two processes both satisfy the atom *critical* at the same time.

Syntax. Fix an infinite set $\text{Vars} = \{i, j, \dots\}$ of vertex variables (called index variables for the clique topology). A *vertex quantifier* is an expression of the form $\exists x : type(x) = m$ or $\forall x : type(x) = m$ where $m \in \mathbb{N}$. An *indexed CTL** formula over vertex variables Vars and atomic propositions AP_{pr} is a formula of the form $Q_1 i_1, \dots, Q_k i_k : \varphi.$, where each $i_n \in \text{Vars}$, each Q_{i_n} is an index quantifier, and φ is a CTL* formula over atomic predicates $\text{AP}_{\text{pr}} \times \text{Vars}$.

The semantics is fully described in the full version of this paper. For 1-ary systems we may write $\forall x$ instead of $\forall x : type(x) = 1$. In the syntax of indexed formulas we may write $type(x) = P_m$ instead of $type(x) = m$. i-CTL* denotes the set of all indexed CTL* sentences, and k-CTL* for the set of all k -indexed formulas in i-CTL*, i.e., formulas with k quantifiers. We similarly define indexed versions of various fragments of CTL*, e.g., i-LTL, k-LTL\X and k-CTL $_d$ \X (k-CTL* formulas with nesting depth of path quantifiers at most d). We write $\overline{P}^G \equiv_{k-\text{CTL}_d^*\setminus X} \overline{P}^{G'}$, if \overline{P}^G and $\overline{P}^{G'}$ agree on all k-CTL $_d$ \X formulas.

Note. The index variables are bound *outside* of all the temporal path quantifiers (A and E). In particular, for an existentially quantified LTL formula to be satisfied there must exist a valuation of the index variables such that ϕ holds for all runs (and not one valuation for each run). Thus this logic is sometimes called prenex indexed temporal logic. Note that if one allows index quantifiers inside the scope

of temporal path quantifiers then one quickly reaches undecidability even for systems with no communication [8].

For the remainder of this paper specifications only come from $i\text{-CTL}^*\backslash X$, i.e., without the next-time operators. It is usual in the context of parameterized systems to consider specification logics without the “next” (X) operator.

2.4 Parameterized Topology, Parameterized System, PMCP, Cutoff

Parameterized Topology \mathcal{G} . An (r -ary) *parameterized topology* \mathcal{G} is a set of r -topologies. Moreover, we assume membership in \mathcal{G} is decidable. Typical examples are the set of r -ary cliques or the set of 1-ary rings.

Parameterized Model Checking Problem. Fix an r -ary parameterized topology \mathcal{G} , a set of r -ary system templates \mathcal{P} , and a set of indexed temporal logic sentences \mathcal{F} . The *parameterized model checking problem (PMCP)* for this data, written $\text{PMCP}_{\mathcal{G}}(\mathcal{P}, \mathcal{F})$, is to decide, given a formula $\varphi \in \mathcal{F}$ and a system template $\overline{P} \in \mathcal{P}$, whether for all $G \in \mathcal{G}$, $\overline{P}^G \models \varphi$. The complexity of the $\text{PMCP}_{\mathcal{G}}(\mathcal{P}, \mathcal{F})$, where the formula $\varphi \in \mathcal{F}$ is fixed and only the system template is given as an input, is called the *program complexity*.

Cutoff. A *cutoff* for $\text{PMCP}_{\mathcal{G}}(\mathcal{P}, \mathcal{F})$ is a natural number c such that for every $\overline{P} \in \mathcal{P}$ and $\varphi \in \mathcal{F}$, the following are equivalent: **(i)** $\overline{P}^G \models \varphi$ for all $G \in \mathcal{G}$ with $|V_G| \leq c$; **(ii)** $\overline{P}^G \models \varphi$ for all $G \in \mathcal{G}$.

Lemma 1. *If $\text{PMCP}_{\mathcal{G}}(\mathcal{P}, \mathcal{F})$ has a cutoff, then $\text{PMCP}_{\mathcal{G}}(\mathcal{P}, \mathcal{F})$ is decidable*

Proof. If c is a cutoff, let G_1, \dots, G_n be all topologies G in \mathcal{G} such that $|V_G| \leq c$. The algorithm that solves PMCP takes \overline{P}, φ as input and checks whether or not $\overline{P}^{G_i} \models \varphi$ for all $1 \leq i \leq n$. \square

2.5 Destuttering and Process Executions

The *destuttering* of an infinite word $\alpha \in \Sigma^\omega$ is the infinite word $\alpha^\delta \in \Sigma^\omega$ defined by replacing every maximal finite consecutive sequence of repeated symbols in α by one copy of that symbol. Thus, the destuttering of $(aab)^{\omega}$ is $(ab)^\omega$; and the destuttering of aab^ω is ab^ω . The *destuttering* of set $L \subseteq \Sigma^\omega$, written L^δ , is the set $\{\alpha^\delta \mid \alpha \in L\} \subseteq \Sigma^\omega$.

It is known that $\text{LTL} \setminus X$ can not distinguish between a word and its destuttering, which is the main motivation for the following definition.

Process Executions. For parameterized r -topology \mathcal{G} , r -ary system template $\overline{P} = (P_1, \dots, P_r)$ and $t \leq r$, define the set of *(process) executions (with respect to $t, \overline{P}, \mathcal{G}$)*, written $t\text{-EXEC}_{\mathcal{G}, \overline{P}}$, as the destuttering of the following set:

$$\bigcup_{G \in \mathcal{G}} \{ \text{proj}_v(\pi) \mid \pi \text{ is a state-labelled run of } \overline{P}^G \text{ and } v \in V_G \text{ is of type } t \}.$$

When \mathcal{G} or \overline{P} is clear from the context we may omit them.

The point is that for universal 1-index $\text{LTL} \setminus X$ we can reduce the PMCP to model checking a single system whose runs are $t\text{-EXEC}_{\mathcal{G}, \overline{P}}$. This is explained in details in the full version of this paper.

2.6 Two prominent kinds of pairwise-rendezvous systems

Identical processes. Concurrent systems in which all processes are identical are modeled with system arity $r = 1$. In this case there is a single process template P , and a topology may be thought of as a directed graph $G = (V, E)$ (formally $G = (V, E, T_1)$ with $T_1 = V$). We write $\text{USER-EXEC}_{\mathcal{G}}(U)$ for the set of executions of the user processes in a 1-ary system, i.e., $1\text{-EXEC}_{\mathcal{G}, U}$.

Identical processes with a controller. Concurrent systems in which all processes are identical except for one process (typically called a controller or the environment) are modeled with system arity $r = 2$, and system templates of the form (P_1, P_2) , and we restrict the topologies so that exactly one vertex has type 1 (i.e., runs the controller). We call such topologies *controlled*. We often write (C, U) instead of (P_1, P_2) , and $G = (V, E, v)$ instead of $(V, E, \{v\}, V \setminus \{v\})$. We write $\text{CONTROLLER-EXEC}_{\mathcal{G}}(C, U)$ for the set of executions of the controller process, i.e., $1\text{-EXEC}_{\mathcal{G}, (C, U)}$. We write $\text{USER-EXEC}_{\mathcal{G}}(C, U)$ for the set of executions of the user processes in this 2-ary system, i.e., $2\text{-EXEC}_{\mathcal{G}, (C, U)}$.

2.7 Classes of parameterized topologies

Here we define the classes of parameterized topologies which we will use in the sequel. The classes we define all have bounded clique-width.

w-terms and clique-width. An r -ary w -topology $(V, E, T_1, \dots, T_r, C_1, \dots, C_w)$ extends (V, E, T_1, \dots, T_r) by a partition (C_1, \dots, C_w) of V . For every $u \in V$, if $u \in C_i$ then we say u has color i . We define the w -terms inductively. ϵ is a w -term. If x, y are w -terms, then $\text{add}_{i,t}(x)$, $\text{recol}_{i,j}(x)$, $\text{edge}_{i,j}(x)$ and $x \sqcup y$ are w -terms for $i, j \in [w]$, $t \in [r]$. Every w -term x has an associated w -topology $[[x]]$:

- $[[\epsilon]]$ has $V = E = \emptyset$ and empty labeling.
- $[[\text{add}_{i,t}(x)]]$ is formed by adding a new vertex of color i and type t to $[[x]]$.
- $[[\text{recol}_{i,j}(x)]]$ is formed by recoloring every vertex with color i of $[[x]]$ by j .
- $[[\text{edge}_{i,j}(x)]]$ is formed from $[[x]]$ by adding an edge from every vertex of color i to every vertex of color j .
- $[[x \sqcup y]]$ is the disjoint union of x and y and the union of the labelings.

A topology G has *clique-width at most w* if there is a w -term ρ such that G is isomorphic to $[[\rho(\epsilon)]]$ (forgetting the coloring C_1, \dots, C_w). Every topology of size n has clique-width at most n . A class of topologies \mathcal{G} has *bounded clique-width* if there exists w such that every graph in \mathcal{G} has clique-width at most w . It is well-known if \mathcal{G} has bounded tree-width, then it has bounded clique-width.

Monadic Second Order Logic MSO. MSO is a powerful logic for graphs and graph-like structures. It is the extension of First Order Logic with set quantification. MSO can define classic graph-theoretic concepts such as planarity, connectivity, c -regularity and c -colorability. We assume the reader is familiar with Monadic Second Order logic as described e.g. in [11]. A parameterized topology \mathcal{G} is *MSO-definable* if there exists an MSO-formula Φ such that $G \in \mathcal{G}$

iff $G \models \varPhi$. E.g., $\exists U \forall x \forall y (E(x, y) \rightarrow (U(x) \leftrightarrow \neg U(y)))$ defines the set of bipartite graphs. We denote by \equiv_q^{MSO} the equivalence relation of topologies of being indistinguishable by MSO-formulas of quantifier rank q .

Theorem 1 (Courcelle's Theorem, see [11]). *Let $w \geq 1$ and let $\varphi \in \text{MSO}$. The MSO theory of r -topologies of clique-width w is decidable. I.e., there is an algorithm that on input $\varphi \in \text{MSO}$, decides whether there is an r -topology G of clique-width at most w such that $G \models \varphi$. Moreover, the number of equivalence classes in \equiv_q^{MSO} is finite and computable, and a topology belonging to each class is computable.*

We now define a user-friendly and expressive formalism that can be used to generate natural parameterized topologies.

Iteratively constructible parameterized topologies. A parameterized topology is *iteratively constructible* if it can be built from an initial labeled graph by means of a repeated fixed succession of elementary operations involving addition of vertices and edges, deletion of edges, and relabeling. More precisely, an r -ary parameterized topology \mathcal{G} is *iteratively-constructible* if there are w -terms $\rho(x), \sigma(x)$ with one variable x and no use of disjoint union, and a w -graph H_0 such that (i) $G \in \mathcal{G}$ iff $G = \sigma(\rho^n(H_0))$ for some $n \in \mathbb{N}$, where $\rho^0(H) = H$, (ii) exactly one vertex of H_0 has type 1, and (iii) no vertex of type 1 is added in ρ or σ . For terms $\rho(\cdot)$ and $\rho'(\cdot)$ we write $\rho :: \rho'$ instead of $\rho(\rho'(\cdot))$. Intuitively, ρ “builds up” the topology, and σ puts on the “finishing touch” (see examples below). The unique vertex of type 1 can act as the controller if it is assigned a unique process template, and it is the initial token position in TPSs.

Example 1 (Cliques and rings). The set of cliques (irreflexive) is iteratively constructible: let H_0 consist of a single vertex v of color 1 and type 1, let $\rho(x)$ be $\text{edge}_{1,1} :: \text{add}_{1,2}(x)$, and $\sigma(x)$ be the identity.

The set of uni-directional rings is iteratively constructible: let H_0 consist of two vertices, one of color 1 and type 1 and one of color 2 and type 2 with an edge from 1 to 2. Let $\rho(x)$ be $\text{recol}_{4,2} :: \text{recol}_{2,3} :: \text{edge}_{2,4} :: \text{add}_{4,2}$ and $\sigma(x) = \text{edge}_{2,1}$.

Clique-like (and controllerless clique-like) parameterized topologies. We now define other sets of topologies of bounded clique-width that generalise cliques and stars, but not rings.

Let H be an r -ary topology with vertex set V_H of size m in which each vertex has a distinct type. Let $\rho_2(x) = \text{add}_{1,\text{type}(1)} :: \dots :: \text{add}_{m,\text{type}(m)}$. Let $\rho_1(x)$ be the m -term obtained by the composition of $\text{edge}_{i,j}$ for all $(i, j) \in E_H$ (in an arbitrary order). Let $\rho(x) = \rho_1(x) :: \rho_2(x)$. We have $[[\rho(\epsilon)]] = H$.

An r -ary parameterized topology \mathcal{G} is *clique-like* if there is an r -ary topology H and a partition $B_{sng}, B_{clq}, B_{ind}$ of V_H such that $G \in \mathcal{G}$ iff there exists a function $\text{num} : B_{clq} \cup B_{ind} \rightarrow \mathbb{N}$ such that $[[\rho^{\text{num}}(\epsilon)]] = G$, and ρ^{num} is obtained from ρ by (i) repeating each $\text{add}_{i,\text{type}(i)}$ $\text{num}(i)$ times rather than once, and (ii) finally performing $\text{edge}_{i,i}$ for all $i \in B_{clq}$. Intuitively, G is obtained from H by substituting each vertex in B_{clq} with a clique, each vertex in B_{ind} with an independent set, and leaving every vertex in B_{sng} as a single vertex.

We say that \mathcal{G} is *generated by* H and $B_{sng}, B_{clq}, B_{ind}$. The cardinality of B_{sng} is the *number of controllers* in \mathcal{G} . In case $B_{sng} = \emptyset$ we say that \mathcal{G} is *controllerless*.

Example. Cliques, stars and complete bipartite graphs. Let H be the 2-topology with vertex set $V_H = \{1, 2\}$ and edge set $\{(1, 2), (2, 1)\}$ and $\text{type}(i) = i$ for $i \in [2]$. The set of 2-ary cliques in which exactly one index has type 1 is clique-like using H as defined, $B_{clq} = \{2\}$, $B_{ind} = \emptyset$ and $B_{sng} = \{1\}$. The set of stars in which exactly one index has type 1 is clique-like using H above, $B_{clq} = \emptyset$, $B_{ind} = \{2\}$ and $B_{sng} = \{1\}$. The set of topologies that are complete bipartite graphs is clique-like using H above, $B_{ind} = \{1, 2\}$, and $B_{clq} = B_{sng} = \emptyset$.

Example. Rings are not clique-like. Clique-like parameterized topologies have diameter at most $|V_H|$ unless their diameter is infinite. Rings have unbounded but finite diameter and are therefore not clique-like.

3 Results for Pairwise-Rendezvous Systems

The known decidability results for parameterized pairwise-rendezvous systems are for clique topologies and specifications from 1-indexed $LTL \setminus X$. [3]. Thus we might hope to generalise this result in two directions: more general specification languages and more general topologies. We first show, by reducing the non-halting problem of two-counter machines (2CMs) to the PMCP, that allowing branching specifications results in undecidability:

Theorem 2. $PMCP_{\mathcal{G}}(\mathcal{P}, \mathcal{F})$ is undecidable where \mathcal{F} is the set of 1-indexed $CTL_2^* \setminus X$ formulas, \mathcal{G} is the set of 1-ary clique topologies, and \mathcal{P} is the set of 1-ary system templates.

We conclude that we should restrict the specification logic if we want decidability. In the rest of this section we focus on 1-indexed $LTL \setminus X$ and parameterized clique-topologies with or without a controller (note that the PMCP for 1-indexed $LTL \setminus X$ is undecidable for topologies that contain uni-directional rings [1,2]).

Pairwise Rendezvous: Complexity of PMCP. The proof of the following theorem extends the technique used in [3, Theorem 3.6] for clique topologies:

Theorem 3. Fix an r -ary clique-like parameterized topology \mathcal{G} , let \mathcal{F} be the set of 1-index $LTL \setminus X$ formulas, and \mathcal{P} the set of r -ary system templates. Then $PMCP_{\mathcal{G}}(\mathcal{P}, \mathcal{F})$ is decidable in EXPSPACE.

Thus, using the fact that PMCP is EXPSPACE-hard already for clique topologies and the coverability problem [10], we get:

Theorem 4. Fix an r -ary clique-like parameterized topology \mathcal{G} , let \mathcal{F} be the set of 1-index $LTL \setminus X$ formulas, and \mathcal{P} the set of r -ary system templates. Then $PMCP_{\mathcal{G}}(\mathcal{P}, \mathcal{F})$ is EXPSPACE-complete. The same holds for program complexity.

It is known that PMCP for 1-ary cliques is PSPACE-complete (the upper bound is from [3, Section 4], and the lower bound holds already for $LTL \setminus X$ model checking a single finite state system P , with no communication). We extend the

upper bound to clique-like topologies in which $B_{sng} = \emptyset$, i.e., controllerless clique-like parameterized topologies. The proof follows [3] and is via a reduction to emptiness of Büchi automata, see Theorem 8.

Theorem 5. *Fix an r -ary controllerless clique-like parameterized topology \mathcal{G} , let \mathcal{F} be the set of 1-index $LTL \setminus X$ formulas, and \mathcal{P} the set of r -ary system templates. Then $PMCP_{\mathcal{G}}(\mathcal{P}, \mathcal{F})$ is PSPACE-complete, and the program complexity is in PTIME.*

Pairwise Rendezvous: Cutoffs.

Theorem 6. *Let \mathcal{G} be the 1-ary parameterized clique topology and let \mathcal{F} be the set of 1-index $LTL \setminus X$ formulas. There exists a process template P such that $PMCP_{\mathcal{G}}(\{P\}, \mathcal{F})$ has no cutoff.*

Proof (Sketch). Define process template $P = (S, R, I, \Phi)$ by $S := \{1, 2, 3\}$, $I = \{1\}$, $R = \{(1, \tau, 1), (1, a!, 2), (2, \tau, 1), (1, a?, 3)\}$, and $\Phi(i) = \{i\}$. Thus in a system with $n + 1$ processes one possible behaviour is, up to stuttering, $(12)^n 1^\omega$. This run does not appear in any system with $\leq n$ processes. Thus take the formula ϕ_n stating that for every process and every path, the initial segment, up to stuttering, is not of the form $(12)^n$ (for instance $1 \wedge (1 \cup (2 \wedge (2 \cup 1)))$ states that there is an initial prefix of the form $11^* 22^* 11^*$). \square

Pairwise Rendezvous: Equivalence to finite-state systems. The following theorem says that if there is a cutoff for the set of 1-indexed $LTL \setminus X$ formulas then the set of executions is ω -regular. The proof uses the fact that 1-indexed $LTL \setminus X$ is expressive enough to describe finite prefixes of infinite words, and deducing that since all finite executions of a system of any size must already appear in systems up to the cutoff size, so do the infinite executions. This holds for general topologies, not only for clique-like ones.

Theorem 7. *Fix r -ary parameterized topology \mathcal{G} , let \mathcal{F} be the set of 1-index $LTL \setminus X$ formulas, and let \bar{P} be an r -ary system template. If $PMCP_{\mathcal{G}}(\{\bar{P}\}, \mathcal{F})$ has a cutoff, then for every $t \leq r$, the set of executions $t\text{-EXEC}_{\mathcal{G}, \bar{P}}$ is ω -regular.*

The following theorem states that the set of executions of each process in a controllerless parameterized clique-like topology is ω -regular, i.e., recognizable by a Non-deterministic Büchi Word automaton (NBW)(see [12] for a definition). This is done by a reduction to the case of a clique topology and using the corresponding result in [3, Section 4]⁶

Theorem 8. *For every controllerless clique-like r -ary parameterized topology \mathcal{G} , every r -ary system template \bar{P} , and every $i \leq r$, there is a linearly sized NBW (computable in PTIME) that recognises the set $i\text{-EXEC}_{\mathcal{G}, \bar{P}}$.*

⁶ The relevant result in [3, Section 4] is correct. However, its proof has some bugs and some of the statements (e.g., Theorem 4.8) are wrong. In the full version of this paper we give a correct proof for the main result of [3, Section 4].

By constructing an appropriate system template, and using a pumping argument, we are able to show that the set of executions of systems with a controller is not, in general, ω -regular. More precisely:

Theorem 9. *Let \mathcal{G} be the 2-ary parameterized clique topology. There exist a system template (C, U) for which $\text{CONTROLLER-EXEC}_{\mathcal{G}}(C, U)$ is not ω -regular.*

4 Results for Disjunctive Guards

In the following we will consider parameterized systems as described in Section 2.6, i.e., with an arbitrary number of copies of one template U , and possibly with a unique controller C , arranged in a clique.

The following theorem follows similar lines as Theorem 2, and uses a reduction from the non-halting problem of 2CMs. The main complication here is that, unlike the case of pairwise rendezvous, mutual exclusion is not easily obtainable using disjunctive guards, and thus more complicated gadgets are needed to ensure that the counter operations are simulated correctly.

Theorem 10. *$\text{PMCP}_{\mathcal{G}}(\mathcal{P}, \mathcal{F})$ is undecidable where \mathcal{F} is the set of 1-indexed $\text{CTL}_2^* \setminus X$ formulae, \mathcal{G} is the 1-ary parameterized clique topology, and \mathcal{P} is the set of 1-ary disjunctively-guarded system templates.*

We conclude that we should restrict the specification logic if we want decidability, and in the rest of this section we focus on 1-indexed $\text{LTL} \setminus X$.

Disjunctive Guards: Cutoffs. By [4], for the r -ary parameterized clique topology and k -indexed $\text{LTL} \setminus X$ formulae, there is a cutoff of size $|U| + 2$ (where U is the process template). The following proposition shows that this cutoff is tight.

Proposition 1. *Let \mathcal{G} be the r -ary parameterized clique topology, let \mathcal{F} be the set of 1-index $\text{LTL} \setminus X$ formulae, and let $k > 0$. There is a disjunctively-guarded system template \mathcal{P} of size $\Theta(k)$ such that $\Theta(k)$ is the smallest cutoff for $\text{PMCP}_{\mathcal{G}}(\mathcal{P}, \mathcal{F})$*

Proof (sketch). We show the case of 1-ary cliques. Similar examples exist for r -ary systems, with or without a controller. Consider the process template: $U = (S_U, R_U, I_U, \Phi_U)$ where $S_U = \{s_1, \dots, s_k\}$, $R_U = \{(s_i, s_i, s_{i+1}) \mid i < k\} \cup \{(s_k, s_k, s_1)\} \cup \{(s_i, \top, s_i) \mid i \leq k\}$, $I_U = \{s_1\}$, and $\Phi_U(s_i) = \{s_i\}$; and the formula $\phi_k = \forall x. \text{AG}((s_k, x) \rightarrow G(s_k, x))$. Evidently, ϕ_k holds in all systems with at most k processes, but false in systems with $k + 1$ or more processes.

Disjunctive Guards: Equivalence to finite-state systems. There are several techniques for solving the PMCP for 1-indexed $\text{LTL} \setminus X$ formulae for systems using disjunctive guards. One such technique consists in finding an NBW that model-checks the set of all possible executions of the system, for any number of copies of user processes U . We begin by showing that in general, such an automaton is necessarily big. We show the following lower bound by encoding the language of palindromes of length $2k$.

Proposition 2. Let \mathcal{G} be the 2-ary parameterized controlled clique topology. For every $k > 0$ there exist a disjunctively-guarded system template (C, U) where the sizes of C and U are $\Theta(k)$ such that the smallest NBW whose language is $\text{CONTROLLER-EXEC}_{\mathcal{G}}(C, U)$ has size at least $2^{\Omega(k)}$.

On the other hand, the cutoff $|U| + 2$ yields an NBW of size $|C| \times |U|^{\Omega(|U|)}$, and since this cutoff is tight, this technique can not yield a smaller NBW. In the following theorem we prove, surprisingly, that there is a smaller NBW, of size $O(|C| \times 2^{|U|})$.

Theorem 11. Let \mathcal{G} be the 2-ary parameterized controlled clique topology. For every disjunctively-guarded system template (C, U) there is an NBW of size $O(|C| \times 2^{|U|})$ recognizing the set $\text{CONTROLLER-EXEC}_{\mathcal{G}}(C, U)$. The same is true for $\text{USER-EXEC}_{\mathcal{G}}(C, U)$.

Intuitively, each state in the NBW pairs the current *controller state* together with a set of *reachable user states*, i.e. sets of states of U that can be reached in some system of finite size, given the actual state of the controller C . In this construction, a state $s \in S_U$ is considered reachable iff it is the target of a sequence of transitions in R_U that (a) are not guarded, or (b) are guarded by other reachable states, or (c) are guarded by the current controller state. The NBW has $O(|C| \times 2^{|U|})$ (abstract) configurations, and it is shown that every path in the NBW can be concretized in some system of some finite size.

Disjunctive Guards: Complexity of PMCP. We inherit the PSPACE-hardness of model-checking $LTL \setminus X$ on a single finite-state system. For the upper bound, the construction in Theorem 11 can be done ‘on-the-fly’

Theorem 12. Let \mathcal{G} be the 2-ary parameterized controlled clique topology or the 1-ary parameterized clique topology. Let \mathcal{F} be the set of 1-index $LTL \setminus X$ formulas, and let \mathcal{P} be the set of disjunctively guarded system templates (of suitable arity). The complexity of $\text{PMCP}_{\mathcal{G}}(\mathcal{P}, \mathcal{F})$ is PSPACE-complete.

We inherit the PTIME program complexity (without controller) from Theorem 8. With a controller, the coNP upper bound results from a fine analysis of Theorem 11, and the coNP-hardness by coding of unsatisfiability (the user processes store an assignment, and the controller verifies it is not satisfying).

Theorem 13. Fix \mathcal{F} to be the set of 1-index $LTL \setminus X$ formulas. If \mathcal{P} is the set of 1-ary disjunctively guarded system templates, and \mathcal{G} is the 1-ary parameterized clique topology, then the program complexity of $\text{PMCP}_{\mathcal{G}}(\mathcal{P}, \mathcal{F})$ is PTIME.

If \mathcal{P} is the set of 2-ary disjunctively guarded system templates, and \mathcal{G} is the 2-ary parameterized controlled clique topology, then the program complexity of $\text{PMCP}_{\mathcal{G}}(\mathcal{P}, \mathcal{F})$ is coNP-complete.

5 Results for Token Passing Systems

Theorem 14. Let \mathcal{G} be a parameterized topology that is either iteratively-constructible, or MSO-definable and of bounded clique-width. Then (i) The problem

$\text{PMCP}_{\mathcal{G}}(\mathcal{P}, i\text{-}\text{CTL}^*\backslash X)$ is decidable; (ii) There is an algorithm that given k and d outputs a cutoff for $k\text{-}\text{CTL}_d^*\backslash X$.

Decidability. We use the *finiteness* and *reduction* properties of $k\text{-}\text{CTL}_d^*\backslash X$ from [6]. The reduction property essentially says that the process templates in $\bar{\mathcal{P}}$ play no role, i.e. we can assume the processes in $\bar{\mathcal{P}}_{topo}$ do nothing except send and receive the token. The only atoms are p_j which indicate that j has the token. In a $k\text{-}\text{CTL}_d^*\backslash X$ formula $Q_1x_1 \dots Q_kx_k. \varphi$, every valuation of the variables x_1, \dots, x_k designates k vertices of the underlying topology G , say $\bar{g} = g_1, \dots, g_k$. The formula φ can only use the atoms p_{g_j} for $g_j \in \bar{g}$. We denote the structures of φ by $G|\bar{g}$ to indicate (1) that the process templates are $\bar{\mathcal{P}}_{topo}$ and (2) that \bar{g} have been assigned to x_1, \dots, x_k by quantification. The finiteness property says that there is a computable finite set $CON_{d,k}$ such that every $G|\bar{g}$ is $\equiv_{k\text{-}\text{CTL}_d^*\backslash X}$ -equivalent to a member of $CON_{d,k}$. We use the details of the construction of $CON_{d,k}$ to show essentially that $\equiv_{k\text{-}\text{CTL}_d^*\backslash X}$ is MSO-definable by reducing the quantification on infinite paths in $k\text{-}\text{CTL}_d^*\backslash X$ to MSO quantification on finite simple paths and cycles. Decidability of PMCP is achieved using the decidability of MSO on classes of parameterized topologies of bounded clique-width (Theorem 1). The decidability of PMCP on iteratively constructible parameterized topologies can be shown by employing methods of similar to [9].

Cutoffs. Cutoffs are derived as the maximal size of a representative topology belonging to a \equiv_q^{MSO} -equivalence class as guaranteed in Theorem 14 and are non-elementary due to the number of equivalence classes. For iteratively-constructible parameterized topologies the cutoffs may be much lower, though there exists a system template $\bar{\mathcal{P}}$, and, for all $k \in \mathbb{N}$, an iteratively constructible parameterized topology \mathcal{G}_k of clique-width at most k and a k -indexed $\text{LTL}\backslash X$ formula φ such that the cutoff of $\text{PMCP}_{\mathcal{G}}(\{\bar{\mathcal{P}}\}, \{\varphi\})$ is $2^{\Omega(\sqrt{k})}$.

6 Discussion and related work

The applicability of the reduction of the PMCP to finitely many classical model checking problems as a technique for solving the PMCP depends on the communication primitive, the specification language, and the set of topologies of the system. The wide-ranging nature of our work along these axes gives us some insights which may be pertinent to system models different from our own:

Decidability but no cutoffs. Theorems 3 and 6 show that it can be the case that, for certain sets of specifications formula, cutoffs do not exist yet the PMCP problem is decidable.

Cutoffs may not be optimal. Proposition 1 and Theorem 11 imply that even in cases that cutoffs exist and are computable, they may not yield optimal algorithms for solving the PMCP.

Formalisms for topologies are useful. Many results in Sections 3 and 5 show that decidability and complexity of PMCP can be extended from concrete examples of sets of topologies such as rings and cliques to infinite classes of

topologies given as user-friendly yet powerful formalisms. The formalisms we study may be useful for other system models.

In the context of cutoffs, it is worth noting that we only considered cutoffs with respect to sets of formulas and process templates. As Theorem 6 shows, there is a parameterized topology \mathcal{G} , and a system template \bar{P} , for which no cutoff exists for the set of 1-indexed $LTL \setminus X$ formulas. Note, however, that if the formula φ is also fixed then a cutoff always exists. Indeed, given $\mathcal{G}, \bar{P}, \varphi$, letting $c := |V_G|$ yields a (minimal) cutoff if we choose G to be the smallest for which $\bar{P}^G \not\models \varphi$, or simply the smallest topology in \mathcal{G} if all topologies in \mathcal{G} satisfy φ . We reserve the question of computing the cutoff in such cases to future work.

As previously discussed, this work draws on and generalises the work in [3] on pairwise rendezvous on cliques, the work in [4] on disjunctive guards on cliques, and the work in [6,5,2] on token-passing systems. There are very few published complexity lower-bounds for PMCP (notable exceptions are [10,13]), and to the best of our knowledge, our lower bounds on the sizes of cutoffs are the first proven non-trivial lower bounds for these types of systems.

References

1. Suzuki, I.: Proving properties of a ring of finite-state machines. *Inf. Process. Lett.* **28** (1988) 213–214
2. Emerson, E.A., Namjoshi, K.S.: On reasoning about rings. *Int. J. Found. Comput. Sci.* **14** (2003) 527–550
3. German, S.M., Sistla, A.P.: Reasoning about systems with many processes. *J. ACM* **39** (1992) 675–735
4. Emerson, E.A., Kahlon, V.: Reducing model checking of the many to the few. In: CADE. Springer (2000) 236–254
5. Clarke, E., Talupur, M., Touili, T., Veith, H.: Verification by network decomposition. In: CONCUR 2004. (2004) 276–291
6. Aminof, B., Jacobs, S., Khalimov, A., Rubin, S.: Parameterized model checking of token-passing systems. In: VMCAI, Springer (2014) 262–281
7. Browne, M.C., Clarke, E.M., Grumberg, O.: Reasoning about networks with many identical finite state processes. *Inf. Comput.* **81** (1989) 13–31
8. John, A., Komarov, I., Schmid, U., Veith, H., Widder, J.: Counter attack on byzantine generals: Parameterized model checking of fault-tolerant distributed algorithms. CoRR **abs/1210.3846** (2012)
9. Fischer, E., Makowsky, J.A.: Linear recurrence relations for graph polynomials. In: Pillars of Computer Science, Springer (2008) 266–279
10. Esparza, J.: Keeping a crowd safe: On the complexity of parameterized verification. In: STACS. (2014)
11. Courcelle, B., Engelfriet, J.: Graph Structure and Monadic Second-Order Logic - A Language-Theoretic Approach. Volume 138 of Encyclopedia of mathematics and its applications. Cambridge University Press (2012)
12. Vardi, M., Wolper, P.: Automata-theoretic techniques for modal logics of programs. *Journal of Computer and System Sciences* **32** (1986) 182–221
13. Schmitz, S., Schnoebelen, P.: The power of well-structured systems. In: CONCUR, Springer (2013) 5–24

First Cycle Games*

Benjamin Aminof

IST Austria
Vienna, Austria
benj@ist.ac.at

Sasha Rubin

IST Austria and TU Wien
Vienna, Austria
srubin@ist.ac.at

First cycle games (FCG) are played on a finite graph by two players who push a token along the edges until a vertex is repeated, and a simple cycle is formed. The winner is determined by some fixed property Y of the sequence of labels of the edges (or nodes) forming this cycle. These games are traditionally of interest because of their connection with infinite-duration games such as parity and mean-payoff games.

We study the memory requirements for winning strategies of FCGs and certain associated infinite duration games. We exhibit a simple FCG that is not memoryless determined (this corrects a mistake in *Memoryless determinacy of parity and mean payoff games: a simple proof* by Björklund, Sandberg, Vorobyov (2004) that claims that FCGs for which Y is closed under cyclic permutations are memoryless determined). We show that $\Theta(n)!$ memory (where n is the number of nodes in the graph), which is always sufficient, may be necessary to win some FCGs. On the other hand, we identify easy to check conditions on Y (i.e., Y is closed under cyclic permutations, and both Y and its complement are closed under concatenation) that are sufficient to ensure that the corresponding FCGs and their associated infinite duration games are memoryless determined. We demonstrate that many games considered in the literature, such as mean-payoff, parity, energy, etc., satisfy these conditions. On the complexity side, we show (for efficiently computable Y) that while solving FCGs is in PSPACE, solving some families of FCGs is PSPACE-hard.

1 Introduction

First cycle games (FCGs) are played on a finite graph by two players who push a token along the edges of the graph until a simple cycle is formed. Player 0 wins the play if the sequence of labels of the edges (or nodes) of the cycle satisfies some fixed cycle property Y , and otherwise Player 1 wins. For instance, if every vertex has an integer priority, the cycle property $Y = \text{cyc-Parity}$ states that the largest priority occurring on the cycle should be even. For a fixed cycle property Y , we write $\text{FCG}(Y)$ for the family of games over all possible arenas with this winning condition. We are motivated by two questions: Under what conditions on Y is every game in $\text{FCG}(Y)$ memoryless determined? What is the connection between FCGs and infinite-duration games?

First cycle games. First, we give a simple example showing that first cycle games (FCGs) are not necessarily memoryless determined, even if Y is closed under cyclic permutations (i.e., even if winning depends on the cycle but not on how it was traversed), contrary to the claim in [2][Page 370]. We then show that, for a graph with n nodes, whereas no winning strategy needs more than $(n - 1)!$ memory (since this is enough to remember the whole history of the game), some FCGs require at least $\Omega(n!)$ memory. To complete the picture, we analyse the complexity of solving FCGs and show that it is PSPACE-complete. More specifically, we show that if one can decide in PSPACE whether a given cycle satisfies the property

*This work is supported by the Austrian Science Fund through grant P23499-N23, and through the RiSE network (S11403-N23, S11407-N23); ERC Start grant (279307: Graph Games); and Vienna Science and Technology Fund (WWTF) grant PROSEED Nr. ICT 10-050.

Y , then solving the games in $\text{FCG}(Y)$ is in PSPACE; and that even for a trivially computable cycle property Y (namely, that the cycle ends with the label 0), solving the games in $\text{FCG}(Y)$ is PSPACE-hard.

First Cycle Games and Infinite-Duration Games. The main object used to connect FCGs and infinite-duration games (such as parity games) is the *cycles-decomposition* of a path. Informally, a path is decomposed by pushing the edges of the path onto a stack; as soon as a cycle is detected in the stack it is popped and output, and the algorithm continues. We then say that a winning condition W (such as the parity or energy winning condition) is Y -greedy on \mathcal{A} if in the game on arena \mathcal{A} and winning condition W , Player 0 is guaranteed to win if he ensures that every cycle in the cycles-decomposition of the play satisfies Y , and Player 1 is guaranteed to win if she ensures that every cycle in the cycles-decomposition does not satisfy Y . We prove a *Transfer Theorem*: if W is Y -greedy on \mathcal{A} , then the winning regions in the following two games on arena \mathcal{A} coincide, and memoryless winning strategies transfer between them: the infinite duration game with winning condition W , and the FCG with winning condition Y .

To illustrate the usefulness of the concept of being Y -greedy, we instantiate the definition to well-studied infinite-duration games: i) the parity winning condition (the largest priority occurring infinitely often is even) is Y -greedy on every arena \mathcal{A} where $Y = \text{cyc-Parity}$, ii) the mean-payoff condition (the mean payoff is at least v) is cyc-MeanPayoff_v -greedy on every arena \mathcal{A} (where $\text{cyc-MeanPayoff}_v =$ average payoff is at least v), and iii) for every arena \mathcal{A} with vertex set V , and largest weight W , the energy condition stating that the energy level is always non-negative starting with initial credit $W(|V| - 1)$ is cyc-Energy -greedy on \mathcal{A} (where $\text{cyc-Energy} =$ the energy level is non-negative).

In order to prove memoryless determinacy of certain FCGs (and related infinite-duration games) we generalise techniques used to prove that mean-payoff games are memoryless determined (Ehrenfeucht and Mycielski [4]). Given a cycle property Y , we first consider the infinite duration games $\text{ACG}(Y)$ (all cycles), and $\text{SCG}(Y)$ (suffix all-cycles). A game in the family $\text{ACG}(Y)$ requires Player 0 to ensure that every cycle in the cycles-decomposition of the play (starting from the beginning) satisfies Y . A game in the family $\text{SCG}(Y)$ requires Player 0 to ensure that every cycle in the cycles-decomposition of *some suffix* of the play satisfies Y . As was done in [4], reasoning about infinite and finite duration games is intertwined – in our case, we simultaneously reason about games in $\text{FCG}(Y)$ and $\text{SCG}(Y)$. We define a property of arenas, which we call Y -unambiguous, and prove a *Memoryless Determinacy Theorem*: a game from $\text{FCG}(Y)$ whose arena \mathcal{A} is Y -unambiguous is memoryless determined. Combining this with the Transfer Theorem above, we also get that if \mathcal{A} is Y -unambiguous, then any game with a winning condition W that is Y -greedy on \mathcal{A} , is memoryless determined¹.

Although checking if an arena is Y -unambiguous may not be hard, it has two disadvantages: it involves reasoning about infinite paths and it involves reasoning about the arena whereas, in many cases, memoryless determinacy is guaranteed by the cycle property Y regardless of the arena (this is the case for example with $Y = \text{cyc-Parity}$). Therefore, we also provide easy to check ‘finitary’ sufficient conditions on Y (namely that Y is closed under cyclic permutations, and both Y and its complement are closed under concatenation) that ensure Y -unambiguity of every arena, and thus memoryless determinacy for all games in $\text{FCG}(Y)$. We demonstrate the usefulness of these conditions by observing that typical cycle properties are easily seen to satisfy them, e.g., cyc-Parity , cyc-MeanPayoff_v , cyc-Energy .

We conclude by noting that, in particular, if Y is closed under cyclic permutations, and both Y and its complement are closed under concatenation, then games with winning condition W are memoryless determined on every arena \mathcal{A} for which W is Y -greedy on \mathcal{A} . As noted above, for many winning

¹Taking Y to be cyc-GoodForEnergy (defined to be that either the energy level is positive, or it is zero and the largest priority occurring is even) and noting that for every arena \mathcal{A} we have: i) \mathcal{A} is Y -unambiguous and, ii) the game in $\text{ACG}(Y)$ over \mathcal{A} is Y -greedy on \mathcal{A} ; we obtain a proof of [3][Lemma 4] that no longer relies on the incorrect result from [2].

conditions W (such as mean-payoff, parity, and energy winning conditions) it is easy to find a cycle property Y satisfying the mentioned closure conditions, and for which W is Y -greedy on the arena of interest. This provides an easy way to deduce memoryless determinacy of these classic games.

Related work. As just discussed, this work extends [4], finds a counter-example to a claim in [2], and supplies a proof of a lemma in [3]. Conditions that ensure (or characterise) which games have memoryless strategies appear for example in [1, 5, 6]. However, all of these deal with infinite duration games and do not exploit the connection to finite duration games.

Due to space limitations, proofs appear in the full version of the article.

2 Definitions

In this paper all games are two-player turn-based games of perfect information played on finite graphs. The players are called Player 0 and Player 1.

Arena An *arena* is a labeled directed graph $\mathcal{A} = (V_0, V_1, E, \mathbb{U}, \lambda)$ where

1. V_0 and V_1 are disjoint sets of vertices of Player 0 and Player 1, respectively; the set of vertices of the arena $V := V_0 \cup V_1$ is non-empty.
2. $E \subseteq V \times V$ is a set of edges with no dead-ends (i.e., for every $v \in V$ there is some edge $(v, w) \in E$);
3. \mathbb{U} is a set of possible labels.
4. $\lambda : E \rightarrow \mathbb{U}$ is a *labeling* function, used by the winning condition.

Typical choices for \mathbb{U} are \mathbb{R} and \mathbb{N} . Games in which vertices are labeled instead of edges can be modeled by ensuring $\lambda(v, w) = \lambda(v, w')$ for all $v, w, w' \in V$. Similarly, games in which vertices are labeled by elements of \mathbb{U}' and edges are labeled by elements of \mathbb{U}'' can be modeled by labeling edges by elements of $\mathbb{U}' \times \mathbb{U}''$. As usual, if $u = e_1 e_2 \dots$ is a (finite or infinite) sequence of edges in the arena, we write $\lambda(u)$ for the string of labels $\lambda(e_1)\lambda(e_2)\dots$.

Plays and strategies A *play* $\pi = \pi_0, \pi_1, \dots$ in an arena is an infinite² sequence over V such that $(\pi_j, \pi_{j+1}) \in E$ for all $j \in \mathbb{N}$. The node π_0 is called the *starting node* of the play. We denote the set of all plays in the arena \mathcal{A} by *plays*(\mathcal{A}). A *strategy* for Player i is a function $S : V^* V_i \rightarrow V$ such that if $u \in V^*$ and $v \in V_i$ then $(v, S(uv)) \in E$. A strategy S for Player i is *memoryless* if $S(uv) = S(u'v)$ for all $u, u' \in V^*, v \in V_i$. A play π is *consistent* with S , where S is a strategy for Player i , if for every $j \in \mathbb{N}$ such that $\pi_j \in V_i$, it is the case that $\pi_{j+1} = S(\pi_0 \dots \pi_j)$. A strategy S for Player i is *generated by a Moore machine* if there exists a finite set M of *memory states*, an *initial state* $m_I \in M$, a *memory update* function $\delta : V \times M \rightarrow M$, and a *next-move function* $\rho : V \times M \rightarrow V$ such that if $u = u_0 u_1 \dots u_l$ is a prefix of a play with $u_l \in V_i$ then $S(u) = \rho(u_l, m_l)$ where m_l is defined inductively by $m_0 = m_I$ and $m_{l+1} = \delta(u_l, m_l)$. A strategy S is *finite-memory* if it is generated by some Moore machine. A strategy S uses memory at most k if it is generated by some Moore machine with $|M| \leq k$. A strategy S uses memory at least k if every Moore machine generating S has $|M| \geq k$.

Games, Winning Conditions, and Memoryless Determinacy A *game* is a pair (\mathcal{A}, O) where $\mathcal{A} = (V_0, V_1, E, \mathbb{U}, \lambda)$ is an arena and $O \subseteq \text{plays}(\mathcal{A})$ is an *objective* (usually induced by the labeling). If either V_0 or V_1 is empty, then the game (A, O) is called a *solitaire game*. A play π in a game (\mathcal{A}, O) is *won by Player 0* if $\pi \in O$, and *won by Player 1* otherwise. A strategy S for Player i is *winning starting from a node* $v \in V$ if every play π that starts from v and is consistent with S is won by Player i .

²For simplicity, we consider plays of both finite and infinite duration games to be infinite. However, in a finite duration game (and thus in any FCG) the winner is determined by a finite prefix of the play, and the moves after this prefix are immaterial.

A *winning condition* is a set $W \subseteq \mathbb{U}^\omega$. If W is a winning condition and \mathcal{A} is an arena, the objective $O_W(\mathcal{A})$ induced by W is defined as follows: $O_W(\mathcal{A}) = \{v_0 v_1 v_2 \dots \in \text{plays}(\mathcal{A}) \mid \lambda(v_0, v_1) \lambda(v_1, v_2) \dots \in W\}$. Here are some standard winning conditions:

- The *parity condition* PARITY consists of those infinite sequences $c_1 c_2 \dots \in \mathbb{N}^\omega$ such that the largest label occurring infinitely often is even.
- For $v \in \mathbb{R}$, the *v -mean-payoff condition* consists of those infinite sequences $c_1 c_2 \dots \in \mathbb{R}$ such that $\limsup_{k \rightarrow \infty} \frac{1}{k} \sum_{i=1}^k c_i$ is at least v .
- The *energy condition* for a given *initial credit* $r \in \mathbb{N}$, written ENERGY(r), consists of those infinite sequences $c_1 c_2 \dots \in \mathbb{Z}^\omega$ such that $r + c_1 + \dots + c_k \geq 0$ for all $k \geq 1$.
- The *energy-parity condition* ENERGY-PARITY(r) is defined as consisting of $(c_1, d_1)(c_2, d_2) \dots \in \mathbb{N} \times \mathbb{Z}$ such that $c_1 c_2 \dots$ is in PARITY and $d_1 d_2 \dots$ is in ENERGY(r).

The (*memoryless*) *winning region* of Player i is the set of vertices $v \in V$ such that Player i has a (memoryless) winning strategy starting from v . A game is *pointwise memoryless for Player i* if the memoryless winning region for Player i coincides with the winning region for Player i . A game is *uniform memoryless for Player i* if there is a memoryless strategy for Player i that is winning starting from every vertex in that player's winning region.

A game is *determined* if the winning regions partition V . A game is *pointwise memoryless determined* if it is determined and it is pointwise memoryless for both players. A game is *uniform memoryless determined* if it is determined and uniform memoryless for both players.

Cycles-decomposition A *cycle* in an arena \mathcal{A} is a sequence of edges $(v_1, v_2)(v_2, v_3) \dots (v_{k-1}, v_k)(v_k, v_1)$.

Define an algorithm that processes a play $\pi \in \text{plays}(\mathcal{A})$ and outputs a sequence of cycles: at step 0 start with empty stack; at step j push the edge (π_j, π_{j+1}) , and if for some k , the top k edges on the stack form a cycle, this cycle is popped and output, and the algorithm continues to step $j+1$. The sequence of cycles output by this algorithm is called the *cycles-decomposition of π* , and is denoted by *cycles*(π). The *first cycle of π* is the first cycle in *cycles*(π). For example, if $\pi = vwvwvs(xyz)^\omega$, then *cycles*(π) = $(w, x)(x, w), (v, w)(w, v), (x, y)(y, z)(z, x), (x, y)(y, z)(z, x), \dots$, and the first cycle of π is $(w, x)(x, w)$. Note that *cycles*(π) is such that at most $|V| - 1$ edges of π do not appear in it (i.e., they are pushed but never popped – like the edge (v, s) in the example above). As we show in the full version, this allows one to reason, for instance, about the initial credit problem for energy games (cf. [3]).

Cycle properties A *cycle property* is a set $Y \subseteq \mathbb{U}^*$, used later on to define winning conditions for games. Here are some cycle properties that we refer to in the rest of the article:

1. Let cyc-EvenLen be those sequences $c_1 c_2 \dots c_k \in \mathbb{U}^*$ such that k is even.
2. Let cyc-Parity be those sequences $c_1 \dots c_k \in \mathbb{N}^*$ such that $\max_{1 \leq i \leq k} c_i$ is even.
3. Let cyc-Energy be those sequences $c_1 \dots c_k \in \mathbb{Z}^*$ such that $\sum_{i=1}^k c_i \geq 0$.
4. Let cyc-GoodForEnergy be those sequences $(c_1, d_1) \dots (c_k, d_k) \in (\mathbb{N} \times \mathbb{Z})^*$ such that either $\sum_{i=1}^k d_i > 0$, or both $\sum_{i=1}^k d_i = 0$ and $c_1 \dots c_k \in \text{cyc-Parity}$.
5. Let cyc-MeanPayoff $_v$ be those sequences $c_1 \dots c_k \in \mathbb{R}^*$ such that $\frac{1}{k} \sum_{i=1}^k c_i \leq v$, for some $v \in \mathbb{R}$.
6. Let cyc-MaxFirst be those sequences $c_1 \dots c_k \in \mathbb{N}^*$ such that $c_1 \geq c_i$ for all $1 \leq i \leq k$.
7. Let cyc-EndsZero be those sequences $c_1 \dots c_k \in \mathbb{N}^*$ such that $c_k = 0$.

If $Y \subseteq \mathbb{U}^*$ is a cycle property, write $\neg Y$ for the cycle property $\mathbb{U}^* \setminus Y$. We isolate two important classes of cycle properties (the first is inspired by [2]):

1. Say that Y is *closed under cyclic permutations* if $ab \in Y$ implies $ba \in Y$, for all $a \in \mathbb{U}, b \in \mathbb{U}^*$.
2. Say that Y is *closed under concatenation* if $a \in Y$ and $b \in Y$ imply that $ab \in Y$, for all $a, b \in \mathbb{U}^*$.

Note that the cycle properties 1-5 above are closed under cyclic permutations and concatenation; and that $\neg\text{cyc-EvenLen}$ is closed under cyclic permutations but not under concatenation.

First Cycle Games (FCGs) Given a cycle property $Y \subseteq \mathbb{U}^*$, and an arena $\mathcal{A} = (V_0, V_1, E, \mathbb{U}, \lambda)$, let the objective $O_{\text{FCG}(Y)}(\mathcal{A}) \subseteq \text{plays}(\mathcal{A})$ be such that $\pi \in O_{\text{FCG}(Y)}(\mathcal{A})$ iff $\lambda(u) \in Y$ where u is the *first cycle* in the cycles-decomposition of π . The family $\text{FCG}(Y)$ of *first cycle games of Y* consists of all games of the form $(A, O_{\text{FCG}(Y)}(\mathcal{A}))$ where \mathcal{A} is an arena with labels in \mathbb{U} . For instance, $\text{FCG}(\text{cyc-Parity})$ consists of those games such that Player 0 wins iff the largest label occurring on the first cycle is even.³

3 Finite Duration Cycle Games (on being first)

In this section we analyse the memory required for winning strategies in first cycle games, and the complexity of solving these games. We begin by correcting a mistake in [2].

Proposition 1. *There exists a cycle property Y closed under cyclic permutations and a game in $\text{FCG}(Y)$ that is not pointwise memoryless determined.*

To see this, consider a game where Player 1 chooses from $\{a, b\}$ and Player 0 must match the choice. This clearly requires Player 0 to have memory. The claim follows by simply encoding this game as a FCG. For example, let the cycle-property Y be cyc-EvenLen, let the vertex set be $\{v_1, v_2, v_3, v_4\}$, let $V_0 = \{v_1\}$, and let the edges be $\{(v_1, v_2), (v_2, v_1), (v_1, v_3), (v_3, v_2), (v_2, v_4), (v_4, v_1)\}$.

We now consider the difference between pointwise and uniform memoryless determinacy of FCGs.

Theorem 1. 1. *Solitaire FCGs are pointwise memoryless determined.*

2. *There is a solitaire FCG that is not uniform memoryless determined.*

3. *If cycle property Y is closed under cyclic permutations, and a game from $\text{FCG}(Y)$ is pointwise memoryless for Player i , then that game is uniform memoryless for Player i .*

Proposition 2. 1. *For a FCG on an arena with n vertices, if Player i wins from v , then every winning strategy for Player i starting from v uses memory at most $(n - 1)!$.*

2. *For every n there exists a FCG on an arena with $3n + 1$ vertices, and a vertex v , such that every winning strategy for Player 0 starting from v uses memory at least $n!$.*

The first item is immediate since $(n - 1)!$ is enough to remember the whole history of the game up to the point a cycle is formed. The proof of the second item is by showing a game where Player 1 can “weave” any possible permutation of n nodes, whereas in order to win Player 0 must remember this permutation. The construction is in the full version of the paper.

Finally, we analyse the complexity of solving FCGs with efficiently computable cycle properties.

Theorem 2. 1. *If Y is a cycle property for which solving membership is in PSPACE, then the problem of solving games in $\text{FCG}(Y)$ is in PSPACE.*

2. *The problem of solving games in $\text{FCG}(\text{cyc-EndsZero})$ is PSPACE-complete.*

³Formally, then, first cycle games are of infinite duration, although the winner is determined after the first cycle appears on the play.

Sketch. For the first item, observe that solving the game amounts to evaluating the finite AND-OR tree obtained by unwinding the arena into all possible plays, up to the point on each play where a cycle is formed; nodes belonging to Player 0 are 'or' nodes, nodes belonging to Player 1 are 'and' nodes, and a leaf is marked by 'true' iff the cycle formed on the way to it is in Y . Since this tree has depth at most n (the size of the arena), and since we assumed membership in Y is in PSPACE, marking the leaves can be done in PSPACE. So evaluating the tree can be done in PSPACE.

For the second item, note that Generalised Geography can be thought of as a first cycle game in which Player i nodes are labeled by i , and $Y = \text{cyc-EndsZero}$. Note that computing Y is computationally trivial, but solving Generalised Geography is PSPACE-hard (see for instance [7][Theorem 8.11]). \square

4 Infinite Duration Cycle Games

4.1 On being greedy

We start by defining two types of infinite duration games called the *All-Cycles* and the *Suffix All-Cycles* games, whose winning condition is derived from Y . Informally, All-Cycles games are games in which Player 0 wins iff all cycles in the cycles-decomposition of the play are in Y , and Suffix All-Cycles Games are games in which Player 0 wins iff all cycles in the cycles-decomposition of *some suffix* of the play are in Y . Formally, for arena $\mathcal{A} = (V_0, V_1, E, \mathbb{U}, \lambda)$ and cycle property $Y \subseteq \mathbb{U}^*$, we define two objectives $O \subseteq \text{plays}(\mathcal{A})$ and corresponding families of games as follows:

1. $\pi \in O_{\text{ACG}(Y)}(\mathcal{A})$:if $\lambda(u) \in Y$ for *all cycles* u in *cycles*(π).
2. $\pi \in O_{\text{SCG}(Y)}(\mathcal{A})$:if *some suffix* π' of π satisfies that $\lambda(u) \in Y$ for *all cycles* u in *cycles*(π').⁴

Define the corresponding families of games:

1. The family $\text{ACG}(Y)$ of *all-cycles games of* Y consists of all games of the form $(\mathcal{A}, O_{\text{ACG}(Y)}(\mathcal{A}))$.
2. The family $\text{SCG}(Y)$ of *suffix all-cycles games of* Y consists of all games of the form $(\mathcal{A}, O_{\text{SCG}(Y)}(\mathcal{A}))$.

Definition 1. Say that a game (\mathcal{A}, O) is Y -greedy if $O_{\text{ACG}(Y)}(\mathcal{A}) \subseteq O$ and $O_{\text{ACG}(-Y)}(\mathcal{A}) \subseteq V^\omega \setminus O$. Say that a winning condition W is Y -greedy on arena \mathcal{A} if the game (\mathcal{A}, O_W) is Y -greedy.

Intuitively, W being Y -greedy on \mathcal{A} means that Player 0 can win the game on arena \mathcal{A} with winning condition W if he ensures that every cycle in the cycles-decomposition of the play is in Y , and Player 1 can win if she ensures that every cycle in the cycles-decomposition of the play is not in Y .

For instance, the winning condition PARITY (the largest priority occurring infinitely often is even) is cyc-Parity-greedy on every arena \mathcal{A} , the v -mean-payoff condition (the lim sup average is at least v) is cyc-MeanPayoff _{v} -greedy on every arena \mathcal{A} , and the energy condition (stating that the energy level is always non-negative starting with initial credit $W(|V| - 1)$, where W is the largest weight and V are the vertices of the arena \mathcal{A}) is cyc-Energy-greedy on \mathcal{A} .

Theorem 3 (Transfer). Let (\mathcal{A}, O) be a Y -greedy game, and let $i \in \{0, 1\}$.

1. The winning regions for Player i in the games (\mathcal{A}, O) and $(\mathcal{A}, O_{\text{FCG}(Y)}(\mathcal{A}))$ coincide.
2. For every memoryless strategy S for Player i starting from v in arena \mathcal{A} : S is winning in the game (\mathcal{A}, O) if and only if S is winning in the game $(\mathcal{A}, O_{\text{FCG}(Y)}(\mathcal{A}))$.

⁴Note that this is *not* the same as saying that $\lambda(u) \in Y$ for all but finitely many cycles u in *cycles*(π). For instance, let Y be the property that the cycle has odd length, and take $\pi := (v_1 v_2 v_1 v_3 v_2 v_4)^\omega$. Note that i) decomposing the suffix π' starting with the second vertex results in all cycles having odd length, and ii) it is not the case that almost all cycles in the cycles-decomposition of π have odd length (in fact, they all have even length).

Corollary 1. *Let W be Y -greedy on arena \mathcal{A} . Then the game (\mathcal{A}, O_W) is determined, and is pointwise (uniform) memoryless determined if and only if the game $(\mathcal{A}, O_{FCG(Y)}(\mathcal{A}))$ is pointwise (uniform) memoryless determined.*

4.2 On being unambiguous

Definition 2. *An arena \mathcal{A} is Y -unambiguous if $O_{SCG(Y)}(\mathcal{A}) \cap O_{SCG(-Y)}(\mathcal{A}) = \emptyset$.*

Lemma 1. *If \mathcal{A} is Y -unambiguous then the game $(\mathcal{A}, O_{SCG(Y)}(\mathcal{A}))$ is Y -greedy.*

Theorem 4 (Memoryless Determinacy). *If arena \mathcal{A} is Y -unambiguous, then the game $(\mathcal{A}, O_{FCG(Y)}(\mathcal{A}))$ is pointwise memoryless determined. If Y is also closed under cyclic permutations, then this game is uniform memoryless determined.*

It is of interest to note that the proof of this theorem is a generalisation of the proof used in [4] for showing memoryless determinacy of mean-payoff games. As in [4], our proof reasons about infinite plays. More specifically, we obtain from Theorem 3 and Lemma 1 that the winning regions of each player in the games $(\mathcal{A}, O_{SCG(Y)}(\mathcal{A}))$ and $(\mathcal{A}, O_{FCG(Y)}(\mathcal{A}))$ coincide, and then go on and use this fact to derive memoryless strategies for the game $(\mathcal{A}, O_{FCG(Y)}(\mathcal{A}))$.

Corollary 2. *Suppose arena \mathcal{A} is Y -unambiguous.*

1. *If (\mathcal{A}, O) is Y -greedy, then the game (\mathcal{A}, O) is pointwise memoryless determined.*
2. *The games $(\mathcal{A}, O_{SCG(Y)}(\mathcal{A}))$ and $(\mathcal{A}, O_{ACG(Y)}(\mathcal{A}))$ are pointwise memoryless determined.*

If in addition Y is closed under cyclic permutations, then these game are uniform memoryless determined.

Proof. For the first item combine Theorems 3 and 4. For the second, use Lemma 1 and the fact that $(\mathcal{A}, O_{ACG(Y)}(\mathcal{A}))$ is always Y -greedy. For the final statement apply Theorem 1 item 3. \square

We now provide a simple sufficient condition on Y — that does not involve reasoning about cycles-decompositions of infinite paths — that ensures that every arena \mathcal{A} is Y -unambiguous:

Theorem 5. *Let $Y \subseteq \mathbb{U}^*$ be a cycle property. If Y is closed under cyclic permutations⁵, and both Y and $\neg Y$ are closed under concatenation, then every arena \mathcal{A} is Y -unambiguous.*

It is easy to check that the following cycle properties satisfy the hypothesis of Theorem 5: cyc-Parity, cyc-Energy, cyc-MeanPayoff_v, and cyc-GoodForEnergy. On the other hand, \neg cyc-EvenLen is not closed under concatenation, whereas cyc-MaxFirst is not closed under cyclic permutations.

We conclude with the main result of this section:

Corollary 3. *Suppose Y is closed under cyclic permutations, and both Y and its complement are closed under concatenation. Then the following games are uniform memoryless determined for every arena \mathcal{A} : (\mathcal{A}, O_W) if W is Y -greedy on \mathcal{A} , $(\mathcal{A}, O_{SCG(Y)}(\mathcal{A}))$, and $(\mathcal{A}, O_{ACG(Y)}(\mathcal{A}))$.*

We believe that Corollary 3 provides a practical and easy way of deducing that many infinite duration games are uniform memoryless determined, as follows: exhibit a cycle property Y that is closed under cyclic permutations and both Y and $\neg Y$ are closed under concatenation, such that the winning condition W is Y -greedy on the arena A of interest. Finding such a Y is usually easy since it is simply a ‘finitary’ version of the winning condition W . For example, uniform memoryless determinacy of parity games, mean-payoff games, and energy-games, can easily be deduced by considering the cycle properties cyc-Parity, cyc-MeanPayoff_v, and cyc-Energy.

⁵It may be worth noting that Y is closed under cyclic permutations iff so is $\neg Y$.

References

- [1] Alessandro Bianco, Marco Faella, Fabio Mogavero & Aniello Murano (2011): *Exploring the boundary of half-positionality.* *Ann. Math. Artif. Intell.* 62(1-2), pp. 55–77. Available at <http://dx.doi.org/10.1007/s10472-011-9250-1>.
- [2] Henrik Björklund, Sven Sandberg & Sergei G. Vorobyov (2004): *Memoryless determinacy of parity and mean payoff games: a simple proof.* *Theor. Comput. Sci.* 310(1-3), pp. 365–378. Available at [http://dx.doi.org/10.1016/S0304-3975\(03\)00427-4](http://dx.doi.org/10.1016/S0304-3975(03)00427-4).
- [3] Krishnendu Chatterjee & Laurent Doyen (2012): *Energy parity games.* *Theoretical Computer Science* 458(0), pp. 49 – 60, doi:10.1016/j.tcs.2012.07.038. Available at <http://www.sciencedirect.com/science/article/pii/S0304397512007475>.
- [4] A. Ehrenfeucht & J. Mycielski (1979): *Positional strategies for mean payoff games.* *International Journal of Game Theory* 8(2), pp. 109–113. Available at <http://dx.doi.org/10.1007/BF01768705>.
- [5] Hugo Gimbert & Wiesław Zielonka (2005): *Games Where You Can Play Optimally Without Any Memory.* In: CONCUR, pp. 428–442. Available at http://dx.doi.org/10.1007/11539452_33.
- [6] Eryk Kopczynski (2006): *Half-Positional Determinacy of Infinite Games.* In: ICALP (2), pp. 336–347. Available at http://dx.doi.org/10.1007/11787006_29.
- [7] Michael Sipser (1997): *Introduction to the theory of computation.* PWS Publishing Company.



Alternating traps in Muller and parity games

Andrey Grinshpun^a, Pakawat Phalitnonkiat^b, Sasha Rubin^{c,*}, Andrei Tarfulea^d

^a Department of Mathematics, Massachusetts Institute of Technology, Cambridge, MA, United States

^b Department of Mathematics, Cornell University, Ithaca, NY, United States

^c IST Austria and TU Vienna, Austria

^d Department of Mathematics, Princeton University, Princeton, NJ, United States



ARTICLE INFO

Article history:

Received 24 November 2010

Received in revised form 23 October 2013

Accepted 26 November 2013

Communicated by A. Fraenkel

Keywords:

Parity games

Muller games

ABSTRACT

Muller games are played by two players moving a token along a graph; the winner is determined by the set of vertices that occur infinitely often. The central algorithmic problem is to compute the winning regions for the players. Different classes and representations of Muller games lead to problems of varying computational complexity. One such class are parity games; these are of particular significance in computational complexity, as they remain one of the few combinatorial problems known to be in NP ∩ co-NP but not known to be in P. We show that winning regions for a Muller game can be determined from the alternating structure of its traps. To every Muller game we then associate a natural number that we call its *trap depth*; this parameter measures how complicated the trap structure is. We present algorithms for parity games that run in polynomial time for graphs of bounded trap depth, and in general run in time exponential in the trap depth.

© 2013 Published by Elsevier B.V.

1. Introduction

A Muller game [13,7] is played on a finite directed graph in which every vertex is given a label, either red or blue. There is a token on an initial vertex and two players, call them Red and Blue, move the token along edges; it is Red's move if the token is on a red vertex, and otherwise it is Blue's move. To determine the winner, a Muller game also contains a collection \mathcal{R} of sets of vertices. One assumes that there are no dead ends and so the play is an infinite walk. At each turn one records the vertex under the token. The winner is determined by the set S of vertices that occur infinitely often; Red wins if S is in \mathcal{R} , and otherwise Blue wins.

Every two-player perfect-information game with Borel winning condition is determined [12]: one of the players has a winning strategy. In particular, every Muller game is determined: either Red or Blue has a winning strategy. To solve a Muller game is to determine for every vertex which player has a winning strategy when play starts from the given vertex. This set of vertices is called that player's *winning region*.

One application of these games is to solve Church's synthesis problem: construct a finite-state procedure that transforms any input sequence letter by letter into an output sequence such that the pair of sequences satisfies a given specification. The modern solution to this problem goes through Muller games [17].

Characterization of Muller games. The first part of this paper (Section 3.1) characterizes the winning region of a Muller game G in terms of a two-player reachability game. The length of this reachability game is a measure of the alternating structure of the traps in G ; we call it the *trap-depth* of G . We briefly explain these concepts.

* Corresponding author.

E-mail addresses: agrinshp@mit.edu (A. Grinshpun), pp287@cornell.edu (P. Phalitnonkiat), sasha.rubin@gmail.com (S. Rubin), [\(A. Tarfulea\)](mailto:tarfulea@princeton.edu).

Muller games admit natural substructures, *Attractors* and *Traps*. The Red-attractor [18] of a subset X of vertices is the set of vertices from which Red can force the token into X ; this may be computed in linear time. A Red-trap [18] is a subset Y of vertices in which Blue may keep the token within Y indefinitely (no matter what Red does); i.e. if the token is in Y , Blue may choose to trap Red in the set Y . It should be evident that the complement of a Red-attractor is a Red-trap. Of course, all notions here (and elsewhere) defined for Red may be symmetrically defined for Blue. Thus we talk of Blue-attractors and Blue-traps.

Now, consider the following game played on the same arena as a Muller game G . The *trap-depth game on G in which Red goes first* (Definition 3.2) proceeds as follows (the traps discussed in the following are all nonempty): Red picks a Blue-trap $X_1 \subseteq V$ (here V are the vertices of the Muller game G) which is winning for Red (i.e. $X_1 \in \mathcal{R}$). Then Blue picks a Red-trap Y_1 in the smaller game induced by X_1 , where Y_1 is winning for Blue (i.e. $Y_1 \notin \mathcal{R}$). Then Red picks a Blue-trap X_2 in the game induced by Y_1 such that X_2 is winning for Red. Red and Blue continue like this, alternately choosing traps. The first player that cannot move (i.e., that cannot find an appropriate nonempty trap) loses. As shown in Theorem 3.4,

Red has a nonempty winning region in the Muller game if and only if Red has a winning strategy in the trap-depth game in which Red goes first.

And if Red has a winning strategy in this trap-depth game, the first move of any winning strategy, X_1 , contains only vertices in Red's winning region of the original Muller game.

Application to parity games. The second part of the paper (Section 4) is algorithmic and applies the characterization of winning regions to a particular class of Muller games, parity games.

A parity game [4] is played on a directed graph with vertices labeled by integers called *priorities*. This game is played between two players, Even and Odd, who move a token along edges. A vertex is called *even* if its priority is even, otherwise it is called *odd*. Even moves when the token is on an even vertex, and Odd moves when the token is on an odd vertex. Play starts from a specific vertex; we assume there are no dead ends in the graph and so a play is an infinite walk. Even wins a play if the largest priority occurring infinitely often is even, otherwise Odd wins the play.

It is evident that parity games may be expressed as Muller games: the set \mathcal{R} consists of all subsets X of vertices in which the largest priority of vertices in X is even.

Parity games are intertwined with a logical problem: the model checking problem for Modal μ -calculus formulas is log-space equivalent to solving parity games [7]. In [15] we see how this work can be applied for software verification. Complexity-wise, the problem is known to be in $\text{NP} \cap \text{co-NP}$ [5], and even $\text{UP} \cap \text{co-UP}$ [9]: one of the few combinatorial problems in that category that is not known to be in P. A randomized algorithm was presented in [3] that runs subexponentially (in the size of the input game) in the worst case. Several prior results obtained polynomial-time algorithms for parity games with fixed bounds on certain structural qualities, such as DAG-width; see for instance [1], [2], [8], and [14]. Our approach is in the same vein, with a novel structural quality given by the trap-depth game.

The algorithmically-minded reader may observe a potential drawback with reinterpreting games as a game of alternating traps. The number of traps in a game can grow exponentially with the size of the game (just take a graph with only self-loops), and what's worse is that we are looking at chains of alternating traps. Nonetheless, we apply the characterization to parity games: say that a graph has *Even trap-depth at most k* if Even can guarantee that, in the trap-depth game in which Even goes first, the game ends in a win for Even within k rounds. Then, despite the previous observation, we present an algorithm $\text{TDA}(G, \sigma, k)$ (here G is a parity game, σ is a player, and k an integer) that runs in time $|G|^{O(k)}$ and, as shown in Theorem 4.1,

returns the largest (possibly empty) set starting with which σ can guarantee a win in at most k moves in the trap-depth game on G .

Note that the definition of trap depth may be applied to Muller games as well, though we do not have an algorithmic application; one might hope that there are particularly efficient algorithms for finding winning vertices in Muller games of small trap depth.

Let's put this all together. Say that a parity game has *trap-depth at most k* if either it has Even trap-depth at most k or Odd trap-depth at most k . In Fig. 1 we exhibit, for every integer k , a parity game with $O(k)$ vertices and edges that has trap-depth exactly k . By the end of the paper we will have algorithmically solved the following problems:

- (1) decide if a given parity game G has trap-depth at most k .
- (2) find a nonempty subset of one of the player's winning region assuming the game has trap depth at most k .

Moreover, these problems can be solved in time $O(mn^{2k-1})$ where n is the number of vertices and m the number of edges of a parity game G .

2. Muller games and parity games

A Muller game $G = (V, V_{\text{red}}, E, \mathcal{R})$ satisfies the following conditions: (V, E) is a directed graph in which every vertex has an outgoing edge, V is partitioned into *red* vertices V_{red} and *blue* vertices $V_{\text{blue}} := V \setminus V_{\text{red}}$, and $\mathcal{R} \subset 2^V$ is a collection

of subsets of V . The Muller game is played between two players, Red and Blue. Red will move when the token is on a red vertex, and otherwise Blue will move. Starting from some vertex v_0 , Blue's and Red's moves result in an infinite sequence of vertices, called a *play*, $P = (v_0, v_1, v_2, \dots)$ where $(v_i, v_{i+1}) \in E$. Taking $\inf(P)$ to be the set of vertices that occur infinitely often in the play, i.e. $v \in \inf(P)$ if and only if there are infinitely many i so that $v_i = v$, we say Red *wins the play* if $\inf(P) \in \mathcal{R}$, and otherwise Blue wins the play.

Take $\sigma \in \{\text{Red, Blue}\}$ (we write $\bar{\sigma}$ for the other player, so if σ is Red then $\bar{\sigma}$ is Blue, and vice versa). A σ -*Strategy* is an instruction giving Player σ 's next move given the current token position and play history. Formally, it is a function whose domain is the set of finite strings of vertices $\{v_0 v_1 \dots v_k : (v_i, v_{i+1}) \in E\}$ and whose range is $N(v_k) := \{v \in V : (v_k, v) \in E\}$, the neighborhood of v_k . A σ -strategy is *winning from vertex v_0* if, for all plays starting at v_0 and for which that strategy is followed whenever it is σ 's turn, the resulting play is winning for σ . Finally, a σ -strategy is *memoryless* if it gives σ 's move while taking into consideration only the current token position; i.e., it is a strategy in which the value on $v_0 \dots v_k$ depends only on v_k . A given memoryless σ -strategy π in a Muller game G induces a subgame H in which we restrict the outgoing edges of any σ vertex to the edge defined by π . It is worth noting that if both players fix a strategy for the game, then the resulting play is completely determined by the starting vertex, since given the current history we can determine which vertex is visited next.

Muller games are determined (since they are Borel we can apply [12], although for the special case of regular games see [7]): starting from any vertex, there is a player that has a winning strategy. Determinacy partitions V into the respective *winning regions* $W_{G,\text{Red}}$ and $W_{G,\text{Blue}}$ (where $v \in W_{G,\sigma}$ if and only if σ has a winning strategy starting from v in G). In contexts where the meaning is clear, we will use W_σ for $W_{G,\sigma}$. It follows easily that for a player σ there is a single strategy that wins starting from any vertex in $W_{G,\sigma}$; such a strategy is called a *winning strategy*. We now introduce various important substructures of Muller games that capture some of the essential concepts of reachability and restriction (see Chapter 2.5 of [7]).

Definition 2.1. A σ -*Trap* is a collection of vertices $X \subseteq G$ where:

$$\forall x \in X \cap V_\sigma \text{ we have } N(x) \subseteq X$$

and

$$\forall x \in X \cap V_{\bar{\sigma}}, \exists y \in X \text{ such that } (x, y) \in E.$$

No σ vertex in X has an outgoing edge leaving the trap, and every $\bar{\sigma}$ vertex in X has at least one outgoing edge that stays in the trap. Consequently, if the token ever enters X , $\bar{\sigma}$ has a strategy through which the token will never leave X , no matter what σ does. It is apparent that W_σ is a $\bar{\sigma}$ -trap.

Notation. We write $\text{Traps}_\sigma(G)$ to denote the set of nonempty σ -traps in G .

Definition 2.2. A σ -*Attractor* of a set of vertices Y is the set of vertices starting from which σ has a strategy that guarantees Y will be reached (after finitely many, possibly 0, steps).

We denote the attractor of a set X in a graph G with respect to a player σ by $\text{Attr}(G, X, \sigma)$, and it is worth noting that the attractor of a set may be computed in time linear in the size of the graph; the algorithm for doing so is presented below [18].

Algorithm 1 $\text{Attr}(G = (V, E, p), X, \sigma)$.

```

1:  $C_{\text{prev}} := \emptyset$ 
2:  $C_{\text{cur}} := X$ 
3: while  $C_{\text{cur}} \neq C_{\text{prev}}$  do
4:    $C_{\text{prev}} := C_{\text{cur}}$ 
5:    $C_{\text{cur}} := C_{\text{prev}} \cup \{v \in V_\sigma : N(v) \cap C_{\text{prev}} \neq \emptyset\} \cup \{v \in V_{\bar{\sigma}} : N(v) \subseteq C_{\text{prev}}\}$ 
6: end while
7: return  $C_{\text{cur}}$ 

```

On each iteration, the σ vertices that have an edge into the part of the attractor that has already been computed are added, and the $\bar{\sigma}$ vertices that have only edges into that part are added. We briefly argue correctness: by induction on the number of iterations, we see that starting anywhere in the computed set, σ has a strategy to reach X , and starting outside the computed set it is easy to see that $\bar{\sigma}$ has a strategy to avoid the computed set indefinitely (every $\bar{\sigma}$ vertex outside the set has some edge that does not enter the set, and every σ vertex outside of it has no edge that enters it), so this does compute the attractor.

Definition 2.3. The *Induced Subgame* of G by X is the Muller game using the vertices $V \cap X$ and the edges $E \cap X^2$; we sometimes refer to this as “ G restricted to X ” and use the notation $G[X]$.

Naturally, $G[X]$ should have no dead ends if it is to be a Muller game. It is apparent that G restricted to a trap X is a Muller game. When X is a trap we use *subtraps* to mean the traps of $G[X]$.

Lemma 2.4. (See [18].) If $X \subseteq W_{G,\sigma}$ then, taking $U = \text{Attr}(G, X, \sigma)$, we have $W_{G,\sigma} = U \cup W_{G[V \setminus U],\sigma}$.

In other words, if we know that σ can win from a set, then we can remove that set's attractor from the graph and just find the winning region for σ in the smaller graph.

Lemma 2.5. (See [18].) If $X \subseteq W_{G,\sigma}$ and X is a σ -trap, then $W_{G[X],\sigma} = X$.

Intuitively, this holds because in the induced game $G[X]$ player σ can continue to use the same winning strategy that σ had in G .

We end the section with the statements of some technical lemmas that will be useful. Their proofs are routine.

Lemma 2.6. (See [18].) If X is a σ -trap in G and Y is a σ -trap in $G[X]$, then Y is a σ -trap in G .

The next lemma states that if we take the σ -attractor of some set Y and are interested in how it intersects with some σ -trap X , then the intersection is contained in the attractor of $X \cap Y$ in the game restricted to X .

Lemma 2.7. (See [18].) If X is a σ -trap in G , Y is a set of vertices, and $S = \text{Attr}(G, Y, \sigma)$, then $X \cap S \subseteq \text{Attr}(G[X], X \cap Y, \sigma)$.

Lemma 2.8. If X is a σ -trap in G and Y is a $\bar{\sigma}$ -trap in G , then $X \cap Y$ is a $\bar{\sigma}$ -trap in $G[X]$.

2.1. Parity games

A *Parity game* $G = (V, E, \rho)$ satisfies the following conditions: (V, E) is a directed graph in which every vertex has an outgoing edge, $v_0 \in V$ denotes a starting vertex, and $p : V \rightarrow \mathbb{Z}$ is a function assigning priorities to the vertices. The parity game is played between two players, Even and Odd, where each player moves the token along a directed edge of G whenever the token is on a vertex of the corresponding parity. We say a vertex is even if it has even priority and odd if it has odd priority. Even's and Odd's moves result in an infinite play: $P = (v_0, v_1, v_2, \dots)$ where $(v_i, v_{i+1}) \in E$. Even wins the play if $\limsup_{i \in \mathbb{N}} p(v_i)$ is even and Odd wins otherwise: i.e., the largest priority that occurs infinitely often determines the winner of the play.

Note that, given a Parity game, we may define the corresponding Muller game by placing v in V_{red} if and only if $p(v)$ is even. Then $S \subseteq V$ has $S \in \mathcal{R}$ if and only if $\max(S)$ is even, and otherwise $\max(S)$ is odd and $S \notin \mathcal{R}$. The corresponding Muller game is then $(V, V_{\text{red}}, E, \mathcal{R})$. Note that a play is winning in the Muller game if and only if it is winning in the parity game.

Not only are Parity games determined, they are *Memorylessly Determined* [4]: for every vertex $v \in V$, exactly one of the two players has a memoryless strategy that guarantees a win starting from v . Moreover, for each player there is a single memoryless strategy which, if followed, will result in a winning play starting from any vertex in that player's winning region; this is called a memoryless winning strategy. Note that Muller games are not memorylessly determined; they may require a strategy that uses some of the play history.

3. The trap-depth game

3.1. Main theorem

As mentioned in the introduction, our main result relies on a characterization stemming from chains of alternating subtraps. Each subtrap represents the decision of the corresponding player to further restrict the token's movement. This goes on until the final restriction leaves one player incapable of preventing a winning play for their opponent. We now formalize this idea. We begin by defining a set of statements related to chains of alternating traps.

Define R_σ to be \mathcal{R} if σ is Red and $2^V \setminus \mathcal{R}$ otherwise. The statement $S \in R_\sigma$ says that if the set of vertices that occurs infinitely often is S , then player σ wins. Recall that $\text{Traps}_\sigma(G)$ is the set of nonempty σ -traps in G . Our boolean statements $\Delta_\sigma(G, k)$ are defined recursively and have three parameters: the player σ , the game G , and the iteration (or depth) number k .

Definition 3.1. For player σ , game G , and integer k , the value of $\Delta_\sigma(G, 0)$ is false. For $k > 0$, the value of $\Delta_\sigma(G, k)$ is true if and only if there exists $X \in \text{Traps}_{\bar{\sigma}}(G)$ such that

- $X \in R_\sigma$, and
- $\forall Y \in \text{Traps}_\sigma(G[X])$ we have $Y \in R_\sigma$ or $\Delta_\sigma(G[Y], k - 1)$.

Each statement $\Delta_\sigma(G, k)$ asserts that σ can restrict the token's movement via a trap X in such a way that if every vertex in the trap occurs infinitely often, player σ wins, i.e. $X \in R_\sigma$ (intuitively then, player $\bar{\sigma}$ must choose to further restrict play) and, no matter how $\bar{\sigma}$ further restricts the token's movement via a subtrap Y , either still $Y \in R_\sigma$ or we have that $\Delta_\sigma(G[Y], k - 1)$ is true. So, in particular, $\Delta_{\text{Red}}(G, 1)$ states that there is a Blue-trap X in G with $X \in \mathcal{R}$ such that every Red-subtrap Y has $Y \in \mathcal{R}$.

The above definitions make it easy to see that the statements make references to natural structures in Muller games, but they can be rather cumbersome to work with, so we present an equivalent but easier to visualize way to think about them.

Definition 3.2. Let G be a Muller game. Define the *Trap-Depth game on G in which σ goes first* as follows: in the beginning of the i th round ($i \geq 1$) there will be some current Muller game G_i . The game starts with $G_1 = G$. In the i th round player σ moves first by choosing a trap $X_i \in \text{Traps}_{\bar{\sigma}}(G_i)$ with $X_i \in R_\sigma$. Player $\bar{\sigma}$ replies by choosing a σ -trap Y_i in the subgame $G_i[Y_i]$, i.e. $Y_i \in \text{Traps}_\sigma(G_i[X_i])$, so that $Y_i \in R_{\bar{\sigma}}$. This completes the i th round. Define $G_{i+1} = G_i[Y_i]$. The first player that has no legal move loses.

In a Muller game, this will terminate in at most $\lceil \frac{n}{2} \rceil$ rounds, as each time a player chooses a trap, a vertex must be removed. If the Muller game is a parity game, then the condition $X \in R_\sigma$ simply states that the largest priority of a vertex in X is of parity σ . For a parity game, the number of rounds is at most $\lceil \frac{|p(V)|}{2} \rceil$, since the size of the largest vertex still in play decreases twice per round. In particular, every play in this game is finite and ends in a win for one of the players. Therefore, the game is determined (i.e. one of the players has a winning strategy).

Lemma 3.3. *The value of $\Delta_\sigma(G, k)$ is true if and only if σ has a strategy that ensures their opponent loses the Trap-Depth game in which σ goes first in at most k rounds (so $\bar{\sigma}$ would lose on or before the $2k$ th move).*

This is easily verified by identifying player moves with the quantifiers in the expression for $\Delta_\sigma(G, k)$. We now state the first main result of this paper; Section 3.2 is devoted to its proof.

Theorem 3.4. *Let G be a Muller game. Then $W_{G,\sigma} \neq \emptyset$ if and only if σ has a winning strategy in the trap-depth game on G in which σ goes first. Moreover, the first move X of a winning strategy for player σ satisfies $X \subseteq W_\sigma$.*

So Player σ has a nonempty winning region in the game G if and only if σ has a winning strategy in the Trap-Depth game in which σ goes first.

Note the following simple corollary:

Corollary 3.5. *The following two statements are equivalent:*

- Parity games can be solved in polynomial time.
- The player with a winning strategy in the trap-depth game described by a parity game can be determined in polynomial time.

This theorem also motivates a new parameter for parity games (the parameter applies to Muller games as well, though we do not have an algorithmic application):

Definition 3.6. The *Trap-Depth* of a parity game G is the minimum integer k such that $\Delta_{\text{Even}}(G, k)$ is true or $\Delta_{\text{Odd}}(G, k)$ is true.

One simple class of parity games that has this property is those with a bounded number of priorities, though having bounded trap-depth is much more general than having a bounded number of priorities.

One can define the σ -trap-depth of G as the minimum integer k (if it exists) such that $\Delta_\sigma(G, k)$ is true; so $W_\sigma \neq \emptyset$ if and only if the σ -trap-depth of G is at most $\lceil \frac{|p(V)|}{2} \rceil$. This upper bound can be achieved, as shown by Fig. 1.

3.2. Proof of Theorem 3.4

3.2.1. Proof for memoryless strategies

We will first prove the characterization of Muller games (the first two sentences of Theorem 3.4) for games in which player σ has a memoryless strategy that wins starting from any vertex in W_σ . Intuitively, traps do not distinguish between memoried and memoryless strategies; we will formalize this intuition and this will allow us to extend the main theorem to all Muller games.

Lemma 3.7. *Let G be a nonempty Muller game with $W_{G,\sigma} = V$, that is in which σ wins starting from any vertex, and π a memoryless winning strategy for σ . Then there is a nonempty $\bar{\sigma}$ -trap T in G such that $W_{G[T],\sigma} = T$, $T \in R_\sigma$, and, if π is followed, then any play starting in T will not leave T (i.e. π does not prescribe leaving T).*

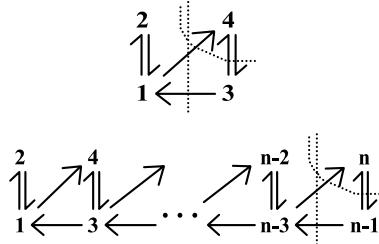


Fig. 1. Maximum trap-depth: Above: base case (G_4) with trap-depth 2; Below: G_n with n vertices (n is even); both are Even-winning from every vertex (so $\Delta_1(G_n, k)$ is never true for any k , by Theorem 3.4). The only Odd-trap is the entire graph, so this must be Even's first move in a trap-depth game. Odd could then remove the right-most top vertex, the remaining set being an Even-subtrap. Within this graph, the only remaining appropriate Odd-subtrap for Even's next move is the set formed by removing the right-most bottom vertex. We have now reduced the game to G_{n-2} . Each time we add two vertices we increase the trap-depth by 1, so the trap depth of G_n is exactly $n/2$.

Proof. Take H to be the subgame induced by π ; that is, leave only one edge out of each σ vertex, the one corresponding to the strategy π . Take T to be a strongly connected component (SCC) of H such that T has no edges into any other SCC. Note that T is a $\bar{\sigma}$ -trap in H , and so also in G . Since T is strongly connected and player σ only has one possible move at any vertex, $\bar{\sigma}$ has a strategy (not necessarily memoryless) such that starting from any vertex in T , if the strategy is followed, every vertex in T occurs infinitely often. Then, by the assumption that π was winning, we must have $T \in R_\sigma$. By construction, π does not prescribe leaving T . \square

The following two propositions establish the theorem for Muller games in which σ has a memoryless winning strategy.

Proposition 3.8. *If $W_{G,\sigma} \neq \emptyset$ and σ has a memoryless winning strategy, then σ has a winning strategy on the trap-depth game on G in which σ goes first.*

Proof. Fix π a memoryless winning strategy for σ . We describe a strategy for σ in the trap-depth game so that for every $i \geq 1$ player σ has a valid move H_i satisfying that π does not prescribe leaving X_i and any potential response Y_i satisfies the invariant $W_{G[Y_i],\sigma} = Y_i$. To get the induction going we define $Y_0 := W_{G,\sigma}$ and note that $W_{G[Y_0],\sigma} = Y_0$. Note that such a strategy ensures that player σ always has a valid move and thus wins the trap-depth game.

Suppose $i \geq 0$ rounds have been played, and assume by induction that σ wins the Muller game starting from any vertex in Y_i . Then, by Lemma 3.7, there is some $\bar{\sigma}$ -trap X_{i+1} in $G[Y_i]$ with $X_{i+1} \in R_\sigma$ such that $W_{G[X_{i+1}],\sigma} = X_{i+1}$ and π does not prescribe leaving X_{i+1} ; have σ play such an X_{i+1} . Then, if player $\bar{\sigma}$ has some response Y_{i+1} , we have that Y_{i+1} is a σ -trap in $G[X_{i+1}]$ and so by Lemma 2.5 $W_{G[Y_{i+1}],\sigma} = Y_{i+1}$, as required. \square

Proposition 3.9. *If $W_{G,\sigma} = \emptyset$ and player $\bar{\sigma}$ has a memoryless winning strategy, then $\bar{\sigma}$ has a strategy that wins the trap-depth game on G in which σ goes first.*

Proof. Let X be player σ 's first move. Then, since X is a $\bar{\sigma}$ -trap, we have $W_{G[X],\bar{\sigma}} = X \neq \emptyset$ by Lemma 2.5. Note that now we simply play the trap-depth game on $G[X]$ in which $\bar{\sigma}$ goes first and $\pi|_X$ is a memoryless winning strategy on $G[X]$, and so by the previous proposition we have that $\bar{\sigma}$ has a winning strategy. \square

The previous two propositions show the desired characterization of Muller games, assuming that players have memoryless winning strategies.

3.2.2. Proof for all Muller games

While Muller games do not, in general, have memoryless strategies, a player need only use a finite amount of memory. To formalize this notion, we define a bounded-state strategy.

Definition 3.10. For any Muller game G , any positive integer N , and any function $M : V \times [N] \rightarrow [N]$, define the M -sequence with respect to any play v_0, v_1, \dots by $M_0 = 0$ and $M_i = M(v_i, M_{i-1})$.

Intuitively, in the above, $M_i \in [N]$ is the (joint) memory used by the players and M_{i+1} depends only on M_i and on the most recent move.

Definition 3.11. For any Muller game G , any positive integer N , and any function $M : V \times [N] \rightarrow [N]$, a strategy π_σ for player σ is a bounded-state M -strategy if there is some $\pi : [N] \rightarrow V$ so that if v_0, \dots is any play consistent with π_σ and M_0, \dots is the corresponding M -sequence, then π_σ depends only on M_i . I.e., there is some function $F\pi : [N] \rightarrow V$ so that for each σ vertex v_i , we have $v_{i+1} = \pi(M_i)$.

The following theorem is proved in [13]. It states that, while Muller games may not have memoryless strategies, players need only a bounded amount of memory.

Theorem 3.12. *For any Muller game G there is some positive integer N and some $M : V \times [N] \rightarrow [N]$ so that, for each player σ , there is a bounded-state M -strategy π_σ satisfying that, starting from any vertex in σ 's winning region, π_σ is a winning strategy for σ .*

Given any Muller game G , take N a positive integer and $M : V \times [N] \rightarrow [N]$ as in the previous theorem. We define the memoried Muller game associated with G , call it G_M , to have vertex set $V \times [N]$ (where N depends on G as in the previous theorem). Intuitively, G_M will simulate G , but each vertex $(v, n) \in V \times [N]$ in the memoried game records the current state of the memory, with v representing the current position in G . Thus, given $(v, n), (w, m)$ vertices in the memoried game, $((v, n), (w, m))$ is an edge of the memoried game if and only if (v, w) is an edge in G and $m = M(w, n)$. Define the vertices belonging to player σ , $V_{\sigma, M}$, by $(v, n) \in V_{\sigma, M}$ if and only if $v \in V_\sigma$. Similarly, $S \subseteq V \times [N]$ is winning for Red, i.e. has $S \in \mathcal{R}_M$, if and only if the corresponding vertices are winning for Red in the original Muller game G , i.e. if and only if $\{v : \exists n, (v, n) \in S\} \in \mathcal{R}$.

Note that by the previous theorem and the construction of the memoried games, both players have memoryless winning strategies in G_M . The remainder of this section argues that the trap-structure of G_M is very similar to that of G .

Intuitively, the following lemma says that if, when playing the trap-depth game on G_M , player $\bar{\sigma}$ simply pretends it's the trap-depth game on G , then any edge out of a $\bar{\sigma}$ vertex that would have existed were the game played on G also exists in the game on G_M .

Lemma 3.13. *Assume, in a trap-depth game on G_M , whenever the current set of vertices is X_M and it is $\bar{\sigma}$'s turn to move, that $\bar{\sigma}$'s move has the following form: taking $X := \{v : \exists n, (v, n) \in X_M\}$, there is some σ -trap Y in $G[X]$ so that $\bar{\sigma}$'s move is $X_M \cap (Y \times [N])$. Then, at every point in the game, if the current set of vertices is X_M , take $X := \{v : \exists n, (v, n) \in X_M\}$. For any $(v, n) \in X_M$ with $v \in V_\sigma$ and for any $w \in X$ so that (v, w) is an edge of G , there is some m so that $((v, n), (w, m))$ is an edge of G_M and $(w, m) \in X_M$.*

Proof. We proceed by induction on the number of plays in the game. In the base case, the game is the whole graph and this is true by construction of G_M . Take X_M to be the current set of vertices and $X := \{v : \exists n, (v, n) \in X_M\}$.

If it is σ 's turn to move, σ chooses some $\bar{\sigma}$ trap Y_M . Taking $Y := \{v : \exists n, (v, n) \in Y_M\}$, for any $(v, n) \in Y_M$ with $v \in V_\sigma$ and for any $w \in X$ so that (v, w) is an edge of G , by induction there is some m so that $((v, n), (w, m))$ is an edge of G_M and $(w, m) \in X_M$. But Y_M is a $\bar{\sigma}$ trap, so since $(v, n) \in Y_M$ and (v, n) is a $\bar{\sigma}$ vertex we get $(w, m) \in Y_M$.

If it is $\bar{\sigma}$'s turn to move, $\bar{\sigma}$ chooses some σ trap Y_M of the form $X_M \cap (Y \times [N])$ where Y is a σ trap in X . Note that $Y = \{v : \exists n, (v, n) \in Y_M\}$, so this notation is consistent with previous notation. Given any $\bar{\sigma}$ vertex $(v, n) \in Y_M$ and any $w \in Y$ so that (v, w) is an edge of G , by induction there must be some m so that $((v, n), (w, m))$ is an edge of G_M and $(w, m) \in X_M$. But then $(w, m) \in Y_M$ since $w \in Y$ and $Y_M = X_M \cap (Y \times [N])$. \square

Theorem 3.14. *Player σ has a winning strategy in a trap-depth game (in which either σ or $\bar{\sigma}$ goes first) on G if and only if player σ has a winning strategy in a trap-depth game on G_M (in which the same player goes first).*

Proof. Assume player σ has a winning strategy in a trap-depth game on G_M . Then player σ is to play a trap-depth game on G and we wish to show that player σ has a winning strategy; we define player σ 's strategy by emulating the game on G_M . With each move, player σ will maintain a set of vertices X_M which represents the state of the emulated game on G_M . Assume the current set of vertices in the game on G is X , and σ has maintained the state X_M . We will inductively show that σ has a strategy that maintains $X = \{v : \exists n, (v, n) \in X_M\}$ and that, starting from X_M with the appropriate player moving, is winning for σ in the game G_M . In the base case, $X = V$ and $X_M = V_M$.

If it is $\bar{\sigma}$'s turn to move, $\bar{\sigma}$ will pick some σ trap $Y \in R_{\bar{\sigma}}$. Then we claim $Y_M := X_M \cap (Y \times [N])$ is a σ -trap in X_M : since Y is a σ -trap in X , it must be the case that given any σ vertex in Y_M , any neighbor it had in X_M is also in Y_M . Given a $\bar{\sigma}$ vertex (v, n) in Y_M , since Y is a σ -trap we have that v has a neighbor w in Y , and so v has a neighbor (w, m) in Y_M by the previous lemma, thus verifying that Y_M is a σ -trap. Then $Y = \{v : \exists n, (v, n) \in Y_M\}$ so $Y_M \in R_{\bar{\sigma}, M}$ since $Y \in R_{\bar{\sigma}}$. Since X_M was winning for σ , we have Y_M is as well (since any move by $\bar{\sigma}$ must result in a winning position).

If it is σ 's turn to move, by assumption we are in some winning position X_M . Then σ may choose some $\bar{\sigma}$ trap $Y_M \in R_{\sigma, M}$ in $G_M[X_M]$ so that Y_M is winning for σ . We claim $Y := \{v : \exists n, (v, n) \in Y_M\}$ is a $\bar{\sigma}$ trap in X . Given any σ vertex $v \in Y$, choose n with $(v, n) \in Y_M$; since Y_M is a $\bar{\sigma}$ trap in X_M , (v, n) must have some neighbor $(w, m) \in Y_M$, so (v, w) is an edge of Y . Given any $\bar{\sigma}$ vertex $v \in Y$ and any neighbor $w \in X$, we may choose n with $(v, n) \in Y_M$; we have that there is some m with $(w, m) \in X_M$ a neighbor of (v, n) by the previous lemma, but Y_M is a $\bar{\sigma}$ -trap, so $(w, m) \in Y_M$ and so $w \in Y$, as desired. Note that, since $Y = \{v : \exists n, (v, n) \in Y_M\}$ and $Y_M \in R_{\sigma, M}$, we get that $Y \in R_\sigma$, so Y is a valid choice of move for player σ .

We've shown that, if σ has a winning strategy on G_M , then σ has a winning strategy on G . Symmetrically, if $\bar{\sigma}$ has a winning strategy on G_M , then $\bar{\sigma}$ has a winning strategy on G , thus proving the theorem. \square

By combining the previous theorem with Propositions 3.8 and 3.9, we may remove the assumptions regarding having memoryless winning strategies:

Theorem 3.15. If $W_{G,\sigma} \neq \emptyset$, then player σ has a winning strategy on the trap-depth game on G in which σ goes first.

Theorem 3.16. If $W_{G,\sigma} = \emptyset$, then player $\bar{\sigma}$ has a winning strategy on the trap-depth game on G in which σ goes first.

Assume T is the first move $\bar{\sigma}$ -trap in the trap-depth game on G where σ goes first and that σ wins starting from $G[T]$ if $\bar{\sigma}$ goes first. If $X := T \cap W_{\bar{\sigma}} \neq \emptyset$, then X is a σ -trap in $G[T]$ with $W_{G[X],\bar{\sigma}} = X$. So, by Lemma 3.7, if we consider X_M in G_M , $\bar{\sigma}$ has a viable move $Y_M \subseteq X_M$ such that $W_{G_M[Y_M],\bar{\sigma}} = Y_M$. By our previous arguments we then get that $\bar{\sigma}$ can win in the trap-depth game in which σ goes first on $G[Y]$ where $Y = \{v_1 : v \in Y_M\}$ is a σ -trap in $G[X]$. Then Y is a valid move for $\bar{\sigma}$ in $G[T]$, contradicting the assumption that σ can win from $G[T]$ if $\bar{\sigma}$ goes first.

Corollary 3.17. If T is the first move of a winning σ -strategy in the Trap-Depth game where σ goes first, then $T \subseteq W_{\sigma}$.

This completes the proof of Theorem 3.4.

It is interesting to understand how these nested traps will interact with modifications to the graph. The following theorem says that via one such modification not much information is lost; this is particularly useful if one wishes to run the algorithms discussed in the next section.

Theorem 3.18. Let G be a Muller game. Assume that, in the trap-depth game on G , X is a valid first move for σ that allows σ to guarantee a win in at most k rounds and that A is a σ -trap so that $A \cap X$ is nonempty. Then there is $Y \subseteq X \cap A$ where Y is a valid first move for σ that allows σ to win in at most k rounds on the trap-depth game on $G[A]$.

Proof. Since X is a $\bar{\sigma}$ trap in G and A is a σ trap in G , we have $X \cap A$ is a $\bar{\sigma}$ trap in $G[A]$. Furthermore, $X \cap A$ is a σ trap in $G[X]$.

If $X \cap A$ is in $R_{\bar{\sigma}}$ then $X \cap A$ is a valid move for $\bar{\sigma}$ in the trap-depth game on G after σ plays X . Therefore, σ must have a response Y that leads to a win in at most $k - 1$ rounds; then Y is a $\bar{\sigma}$ trap in $X \cap A$ and therefore also in A , so it is a valid first move for σ in $G[A]$.

Otherwise, $X \cap A$ is in R_{σ} . We will make $X \cap A$ player σ 's first move in the trap-depth game on $G[A]$. Assume $\bar{\sigma}$ has a response X' so that σ cannot win from X' in at most $k - 1$ rounds. Then X' is a σ trap in $G[X \cap A]$ and therefore also in $G[X]$, so X' is a valid response for $\bar{\sigma}$ in the trap-depth game on G to the play X , contradicting the assumption. \square

Finally, in preparation for the next section, we translate the above into the language of parity games.

Define the “max” of a set of vertices to be those vertices in the set with maximum priority. Recall that $Traps_{\sigma}(G)$ is the set of nonempty σ -traps in G . Then the condition $X \in R_{\sigma}$ becomes $\text{max}(X) \subseteq V_{\sigma}$. For example, we may rewrite the statements Δ :

$$\begin{aligned}\Delta_{\sigma}(G, 0) &:= \text{FALSE}; \\ \Delta_{\sigma}(G, k+1) &:= [\exists X \in Traps_{\bar{\sigma}}(G) \text{ such that } \text{max}(X) \subseteq V_{\sigma}] \text{ and} \\ &\quad [\forall Y \in Traps_{\sigma}(G[X]) \text{ we have } (\Delta_{\sigma}(G[Y], k) \text{ or } \text{max}(Y) \subseteq V_{\sigma})].\end{aligned}$$

In the definition of trap-depth game, for example, when it is player σ 's turn, player σ will choose a $\bar{\sigma}$ trap whose largest priority is of parity σ .

Recall for a parity game G that $TDA(G, \sigma, k)$ returns the largest set X which, as a first move for σ , allows σ to win in at most k rounds in the trap-depth game on G . Theorem 3.18 tells us that the k th trap-depth algorithm is robust in the following sense: if one determines that some vertices are winning for σ and removes their attractor from the graph, either one removes all of $TDA(G, \sigma, k)$ or else one can find the rest of $TDA(G, \sigma, k)$ by repeatedly running the k th trap-depth algorithm on the remaining set.

4. Trap-Depth Algorithms for parity games

In this section of the paper, all discussions are regarding parity games. We present a collection of algorithms that return subsets of the vertices of a parity game, culminating in the Trap-Depth Algorithm (TDA). We will have two versions of TDA (which take different inputs). We will discuss the first of the algorithms later. The characterization of the second algorithm, $TDA(G, \sigma, k)$, follows easily from the first, and it takes as its inputs a parity game G , a player σ , and an integer k . We will ultimately show the following characterization of the second TDA algorithm:

Theorem 4.1. $TDA(G, \sigma, k)$ returns the largest (possibly empty) set S so that if σ uses S as a first move, σ can guarantee a win in at most k moves in the trap-depth game on G .

Note that, by Theorem 3.4, this implies in particular that $TDA(G, \sigma, k) \subseteq W_{\sigma}$.

4.1. Büchi games

Due to the complexity of the TDA, we will first introduce some simpler algorithms. Understanding the simpler algorithms will help significantly in understanding the TDA. The overall structure of the TDA resembles that of a classical algorithm for solving Büchi games, which we present here. A *simple Büchi game*¹ is a parity game in which all of the priorities are either 0 or 1. In the discussions that follow, we will simply say Büchi games; note that the discussions do apply to Büchi games as well as simple Büchi games. Odd wins a Büchi game if and only if Odd has a strategy that reaches vertices of priority 1 infinitely many times. The algorithm takes as input a Büchi game G and returns the winning region for Odd.

Algorithm 2 $\text{Büchi}(G = (V, E, p))$.

```

1:  $\sigma := \text{Odd}$ 
2:  $T_{\text{prev}} := \emptyset$ 
3:  $T_{\text{cur}} := V_\sigma$ 
4: while  $T_{\text{cur}} \neq T_{\text{prev}}$  do
5:    $T_{\text{prev}} := T_{\text{cur}}$ 
6:    $C := \text{Attr}(G, T_{\text{prev}}, \sigma)$ 
7:    $T_{\text{cur}} := T_{\text{prev}} \setminus \{v \in V_\sigma : N(v) \cap C = \emptyset\}$ 
8: end while
9: return  $C$ 
```

The classical algorithm for Büchi games begins each iteration of its while-loop with a set of target vertices $T_{\text{prev}} \subseteq V_{\text{Odd}}$. It then computes the attractor of T_{prev} . This provides the largest set of vertices from which Odd has a strategy for reaching the set T_{prev} . The attractor by itself, however, does not provide any strategy for σ after the token reaches T_{prev} . To remedy this, the algorithm then tests each vertex in T_{prev} to check if it has the option to continue this strategy by returning the token back to the attractor of T_{prev} . If not, then that vertex is removed from being a target. This process repeats until the set T_{prev} stabilizes.

We will first observe that the algorithm outputs a subset of Odd's winning region. To see this, take C to be the output of the algorithm and T to be the final set of target vertices (so that $C = \text{Attr}(G, T, \text{Odd})$). Consider the following strategy for Odd: from any vertex of T , Odd chooses to enter C (this is possible by the termination condition of the algorithm). From any vertex of $C \setminus T$, Odd follows a strategy to reach T . This guarantees that the vertices of T are visited infinitely often; since these vertices have priority 1, this is a winning strategy for Odd.

Conversely, Even has a winning strategy from any vertex not in C . To see this, let T_0, T_1, \dots be the sequence of values of T_{cur} at the beginning of the while-loop in the execution of $\text{Büchi}(G)$ (with the final value repeated). Take $C_i = \text{Attr}(G, T_i, \text{Odd})$. Note that T_i , and therefore C_i , is a decreasing sequence. We claim that any odd vertex in C_i must be in T_i . If some Odd vertex w were in C_i but not in T_i , then there must be some edge from w into C_i . However, w must have been removed from T_j for some $j < i$, which means there is no edge from w into C_j . However, the sequence of C_j is decreasing, a contradiction.

We now proceed by induction to show that Even has a winning strategy from any vertex not in C_i . Note that $T_0 = V_{\text{Odd}}$. Therefore, any vertex not in C_0 is not in the attractor of V_{Odd} , so from any such vertex Even has a strategy that never visits any vertex of priority 1. Take for an inductive hypothesis that, for some i , any vertex not in C_i is winning for Even. Then $T_{i+1} = T_i \setminus \{v \in V_{\text{Odd}} : N(v) \cap C_i = \emptyset\}$. Assume for contradiction that there is some vertex v that is not in C_{i+1} so that v is winning for Odd. Then v must be in $C_i \setminus C_{i+1}$, as otherwise v is winning for Even by the inductive hypothesis. Since v is not in C_{i+1} , Even has a strategy starting from v that avoids T_{i+1} indefinitely. Then consider Even playing the following strategy: as long as the token remains in C_i , Even plays any strategy that avoids entering T_{i+1} . If the token ever leaves C_i , then Even has a winning strategy and uses it. Since v is winning for Odd, Odd must have some winning strategy. Consider the play induced by Even playing the aforementioned strategy and Odd playing a winning strategy. This play cannot leave C_i , as otherwise Even wins. However, some vertex $w \in C_i$ of priority 1 must be visited. Since Even avoids T_{i+1} indefinitely, we must have that w is not in T_{i+1} . Since all Odd vertices in C_i are in T_i , we therefore have that w is in $T_i \setminus T_{i+1}$. However, by definition of T_{i+1} , this means that there are no edges from w into C_i , so the next vertex in the play is not in C_i , a contradiction. This completes the proof of the correctness of the classical Büchi games algorithm.

4.2. $k = 1$

The algorithm for solving trap-depth 1 parity games closely resembles that for Büchi games. The main difference is that, rather than taking the attractor of the target set, we take a “safe” version of the attractor. This takes a parameter λ ; the

¹ This is actually a simpler problem than Büchi games. The way we defined parity games, the player that chooses where to move the token from a given vertex v is just based on the parity of $p(v)$. If we had instead defined parity games in such a way that the player who chooses where to move the token from v may depend on v itself (rather than just on $p(v)$), then parity games in which all priorities are either 0 or 1 would be Büchi games. Under our simplified definition of Büchi games, they may be solved by simply determining if the vertices labeled with 0 contain a cycle. However, the algorithm for solving Büchi games is instructive, even when applied to the simplified Büchi games, so we present it here.

λ -safe attractor in G of a set X for player σ is the set of vertices from which σ has a strategy that guarantees X will be reached and that, in the process, no vertices (excluding those in X) of priority at least λ are visited.

Algorithm 3 SafeAttr($G = (V, E, p), \lambda, X, \sigma$).

```

1:  $C_{\text{prev}} := \emptyset$ 
2:  $C_{\text{cur}} := X$ 
3: while  $C_{\text{cur}} \neq C_{\text{prev}}$  do
4:    $C_{\text{prev}} := C_{\text{cur}}$ 
5:    $C_{\text{cur}} := C_{\text{prev}} \cup \{v \in V_\sigma : p(v) < \lambda \wedge N(v) \cap C_{\text{prev}} \neq \emptyset\} \cup \{v \in V_{\bar{\sigma}} : p(v) < \lambda \wedge N(v) \subseteq C_{\text{prev}}\}$ 
6: end while
7: return  $C_{\text{cur}}$ 
```

At each iteration of the “while” loop, the set C_{cur} (initially X) is enlarged by adding any vertices (of priority less than λ) in V_σ or in $V_{\bar{\sigma}}$ that, respectively, have an edge going into C_{cur} or have only edges going into C_{cur} . Note the similarities to the attractor, [Algorithm 1](#).

Indeed, one sees that $\text{SafeAttr}(G, \lambda, X, \sigma) = \text{Attr}(G, X, \sigma)$ if $\lambda \geq p(\max(V \setminus X))$. And, just like the regular Attractor, one sees that the Safe Attractor stabilizes its own output; i.e., $\text{SafeAttr}(G, \lambda, \text{SafeAttr}(G, \lambda, X, \sigma), \sigma) = \text{SafeAttr}(G, \lambda, X, \sigma)$.

However, it is not obvious how to directly substitute the safe attractor into [Algorithm 2](#), as there does not appear to be a canonical choice for the parameter λ . This motivates the Sequential Safe Attractor algorithm, which, in the trap-depth $k = 1$ case, iteratively applies the λ -safe attractor to σ vertices of priority at least λ . Recall that we defined the max of a set of vertices to be the vertices of largest priority in that set. Below, if S is a set of vertices that all have the same priority, then $p(S)$ is that priority (rather than the singleton containing that priority).

Algorithm 4 SeqAttr₁($G = (V, E, p), X, \sigma$).

```

1:  $W := V_\sigma \cap X$ 
2:  $C := \emptyset$ 
3: while  $W \neq \emptyset$  do
4:    $S := \max(W)$ 
5:    $C := \text{SafeAttr}(G, p(S), C \cup S, \sigma)$ 
6:    $W := W \setminus C$ 
7: end while
8: return  $C$ 
```

At the beginning of the “while” loop above, we have a set C and a list W of target vertices to process. Each iteration of the loop calls the Safe Attractor Algorithm. The Sequential Attractor removes the issue of a priority bound inside the Safe Attractor. For any vertex $v \in V$, SeqAttr₁(G, X, σ) tests if σ has a strategy to move the token from v towards some $w \in X$ in which any resulting path did not visit any vertices of priority at least $p(w)$.

The algorithm for solving trap-depth 1 parity games, TDA₁(G, σ), simply substitutes the Sequential Safe Attractor for the Attractor in the Büchi games algorithm, [Algorithm 2](#).

Algorithm 5 TDA₁($G = (V, E, p), \sigma$).

```

1:  $T_{\text{prev}} := \emptyset$ 
2:  $T_{\text{cur}} := V_\sigma$ 
3: while  $T_{\text{cur}} \neq T_{\text{prev}}$  do
4:    $T_{\text{prev}} := T_{\text{cur}}$ 
5:    $C := \text{SeqAttr}_1(G, T_{\text{prev}}, \sigma)$ 
6:    $T_{\text{cur}} := T_{\text{prev}} \setminus \{v \in V_\sigma : N(v) \cap C = \emptyset\}$ 
7: end while
8: return  $C$ 
```

TDA₁(G, σ) returns the largest $\bar{\sigma}$ trap X in G so that every σ -subtrap Y has that the vertices of largest priority in Y belong to player σ . We will sketch a proof of this fact; a complete proof will follow from the arguments in the next section. The proof will argue three different points, from which the characterization of TDA₁(G, σ) follows immediately:

- (1) (Monotonicity): If A is a $\bar{\sigma}$ -trap in G , then we have $\text{TDA}_1(G[A], \sigma) \subseteq \text{TDA}_1(G, \sigma)$.
- (2) (Completeness): If V satisfies that every σ -trap in V has a vertex whose maximum priority is of parity σ , then $\text{TDA}_1(G = (V, E, p), \sigma) = V$.
- (3) (Soundness): $\text{TDA}_1(G, \sigma)$ is a $\bar{\sigma}$ -trap in G whose maximum priority is of parity σ and which satisfies that every σ -subtrap has maximum priority σ .

To argue monotonicity, one first argues that the Safe Attractor Algorithm is monotonic with respect to its parameter λ , its input set X , and sometimes with respect to the parity game.

Explicitly, if $\lambda_1 \leq \lambda_2$, if $X_1 \subseteq X_2$, and if A is a $\bar{\sigma}$ -trap in G , then

$$\text{SafeAttr}(G[A], \lambda_1, X_1, \sigma) \subseteq \text{SafeAttr}(G, \lambda_2, X_2, \sigma).$$

This is intuitive, but will be proven carefully in the next section.

From this, monotonicity of SeqAttr_1 easily follows. If $X_1 \subseteq X_2$ and if A is a $\bar{\sigma}$ -trap in G , then $\text{SeqAttr}_1(G[A], X_1, \sigma) \subseteq \text{SeqAttr}_1(G, X_2, \sigma)$. A generalization of this will be proven in the next section.

Finally, from this, point (1), monotonicity of TDA_1 , easily follows. Again, a generalization is carefully proved in the next section.

Now we will argue completeness. Assume the whole vertex set V satisfies that every σ -trap in V has maximum priority of parity σ . Then we claim that $\text{SeqAttr}_1(G, V, \sigma) = V$. This will imply that TDA_1 terminates after the first call to SeqAttr and returns V , as desired. To see that $\text{SeqAttr}_1(G, V, \sigma) = V$, consider the first iteration of the “WHILE” loop: since V is a σ -trap in V , the first iteration computes the σ -safe-attractor of $\max(V)$ with $\lambda = p(\max(V))$. However, since these are the vertices of maximum priority, this is the same as computing the attractor. The remaining set is a σ -trap in V , and so has maximum priority of parity σ . By induction, this continues until W is empty and SeqAttr_1 returns all of V .

Finally, we argue the soundness of TDA_1 . At the end of the execution of TDA_1 , there is some final set of target vertices T satisfying every vertex of T has an edge into $\text{SeqAttr}_1(G, T, \sigma)$. By our observations about SeqAttr_1 , starting from any vertex in $\text{TDA}_1(G, \sigma)$, σ has some strategy to reach some vertex w in T so that along the way only priorities less than $p(w)$ are visited. Furthermore, by the terminating condition of TDA_1 , every vertex in T has an edge back into $\text{TDA}_1(G, \sigma)$. This gives that $\text{TDA}_1(G, \sigma)$ is indeed a $\bar{\sigma}$ -trap. Furthermore, starting from the largest vertex in any σ -trap, σ may follow the strategy that allows σ to reach some vertex w in T without seeing larger vertices along the way; σ cannot force the play to leave the trap, and therefore the largest priority in the trap must be w , a σ vertex, as desired.

4.3. General Trap-Depth Algorithm

The main difference between the general Trap-Depth Algorithm (TDA) and TDA_1 is that we change the call to Safe Attractor in TDA_1 to a stronger algorithm, the Generalized Safe Attractor. Indeed, we prove a more general result than [Theorem 4.1](#), allowing us to strengthen any algorithm that satisfies certain conditions.

Definition 4.2. Let $\text{ParAlg}(G, \sigma)$ be an algorithm that takes as input a parity game G and a player σ and returns a subset of the vertices of G . We say ParAlg is *nice with traps* if:

- For any parity game G and any $\bar{\sigma}$ trap A in G ,

$$\text{ParAlg}(G[A], \sigma) \subseteq \text{ParAlg}(G, \sigma).$$

- For any parity game G , taking $S = \text{ParAlg}(G, \sigma)$, for any $\bar{\sigma}$ -trap A containing S , $\text{ParAlg}(G[A], \sigma) = S$, and for any σ -trap A intersecting S , $\text{ParAlg}(G[A], \sigma)$ is nonempty.
- $\text{ParAlg}(G, \sigma)$ always returns a $\bar{\sigma}$ -trap whose largest priority belongs to player σ .

If ParAlg is nice with traps, then we can strengthen it via the TDA. The following definition defines what it means for a set of vertices X to be good; the TDA will return the largest good set of vertices.

Definition 4.3. Let $\text{ParAlg}(G, \sigma)$ be an algorithm that takes as input a parity game G and a player σ and returns a subset of the vertices of G . Given a parity game G and a set of vertices X , we say X is *good* for ParAlg with respect to player σ if X is a $\bar{\sigma}$ -trap whose maximum priority is of parity σ so that for every σ -subtrap Y , either the maximum priority of Y belongs to σ or we have $\text{ParAlg}(G[Y], \sigma)$ is nonempty.

When the context is clear, we will simply say that X is good.

Theorem 4.4. Let $\text{ParAlg}(G, \sigma)$ be an algorithm that takes as input a parity game G and a player σ and returns a subset of the vertices of G . If ParAlg is nice with traps, then $\text{TDA}(G, \sigma, \text{ParAlg})$ returns the largest set of vertices X which is good.

Recursively applying [Theorem 4.4](#) will give [Theorem 4.1](#).

We said that TDA is obtained from TDA_1 by replacing the call to safe attractor, so let us first present the Generalized Safe Attractor Algorithm. The idea behind the safe attractor is to guarantee reaching a target set of vertices in a λ -safe way, so that along the way we don't see vertices of priority at least λ . The idea behind the generalized safe attractor is to guarantee that, if σ fails to reach a target set of vertices, then σ wins the play; in both cases, the only vertices of priority at least λ that are visited are in the target set. We informally refer to a strategy that avoids vertices of priority at least λ as “ λ -safe”.

In order to ensure that everything that is done is λ -safe, we will at some point remove all vertices of priority at least λ from the game by taking the restriction to vertices of lower priority: define the λ -restriction of a parity game $\text{Restrict}(G = (V, E, p), \lambda, \sigma) := V \setminus \text{Attr}(G, \{v \in V : p(v) \geq \lambda\}, \bar{\sigma})$. In words, the only vertices that remain in $\text{Restrict}(G, \lambda, \sigma)$ are those from which σ can ensure that all the priorities in any resulting play are less than λ .

We are now ready to introduce the Generalized Safe Attractor. It takes as input a parity game G , a number λ , a set of target vertices X , a player σ , and an algorithm $\text{ParAlg}(G, \sigma)$. It is most useful to think of the context where $\text{ParAlg}(G, \sigma)$ is nice with traps and always returns a subset of σ 's winning region.

Algorithm 6 $\text{GenAttr}(G = (V, E, p), \lambda, X, \sigma, \text{ParAlg})$.

```

1:  $C_{\text{prev}} := \emptyset$ 
2:  $C_{\text{cur}} := X$ 
3: while  $C_{\text{cur}} \neq C_{\text{prev}}$  do
4:    $C_{\text{prev}} := C_{\text{cur}}$ 
5:    $S := \text{SafeAttr}(G, \lambda, C_{\text{prev}}, \sigma)$ 
6:    $V' := \text{Restrict}(G[V \setminus S], \lambda, \sigma)$ 
7:    $C_{\text{cur}} := S \cup \text{ParAlg}(G[V'], \sigma)$ 
8: end while
9: return  $C_{\text{cur}}$ 
```

At the general step of the “while” loop, we begin with a set C_{prev} of vertices which we want to reach in a λ -safe way. The loop then calls the Safe Attractor Algorithm. $\text{SafeAttr}(G, \lambda, C_{\text{prev}}, \sigma)$ returns the largest collection of vertices S from which σ has a strategy to force the token into C_{prev} such that the token only hits vertices of priority smaller than λ along the way. Once the set S has been found, we check if, given that $\bar{\sigma}$ avoids X , player σ has any *winning* set of vertices given by ParAlg (which is λ -safe) on the remaining set $V \setminus S$; we add this to S to get C_{cur} . Each iteration either adds vertices to C_{cur} or terminates the loop. Since $|C_{\text{cur}}|$ cannot increase indefinitely, GenAttr eventually halts.

For a vertex v and a subset X , v will be in GenAttr if and only if there is a σ -strategy to move the token from v towards X such that, depending on $\bar{\sigma}$'s moves, *either* the token eventually reaches X *or* the token reaches a vertex that ParAlg guarantees is winning for σ ; in both cases, all the vertices visited by the token have priority less than λ , except perhaps the ones in X .

As the name suggests, the Generalized Safe Attractor is a generalization of the Safe Attractor. Consider the case where $\text{ParAlg}(G, \sigma)$ simply returns the empty set for every input. Under this condition, we claim that $\text{GenAttr}(G, \lambda, X, \sigma, \text{ParAlg}) = \text{SafeAttr}(G, \lambda, X, \sigma)$. Later we will show that SafeAttr stabilizes its own output; this immediately gives that, if ParAlg is always empty, a call to GenAttr will have at the end of the first “WHILE” loop C_{cur} equal to SafeAttr , and will subsequently terminate with this output.

Both the general case of the sequential safe attractor algorithm and the TDA are analogous to the ones before, except the sequential safe attractor calls the generalized safe attractor. As before, if S is a set of vertices that all have the same priority, then we write $p(S)$ to denote that priority.

Algorithm 7 $\text{SeqAttr}(G = (V, E, p), X, \sigma, \text{ParAlg})$.

```

1:  $W := V_\sigma \cap X$ 
2:  $C := \emptyset$ 
3: while  $W \neq \emptyset$  do
4:    $S := \max(W)$ 
5:    $C := \text{GenAttr}(G, p(S), C \cup S, \sigma, \text{ParAlg})$ 
6:    $W := W \setminus C$ 
7: end while
8: return  $C$ 
```

For any $v \in \text{SeqAttr}(G, X, \sigma, \text{ParAlg})$, σ has a strategy to ensure that, if the token ever reaches some $w \in X$, it hits only vertices of priority smaller than $p(w)$ along the way and, if it never reaches X , then σ wins.

TDA simply calls the sequential safe attractor.

Algorithm 8 $\text{TDA}(G = (V, E, p), \sigma, \text{ParAlg})$.

```

1:  $T_{\text{prev}} := \emptyset$ 
2:  $T_{\text{cur}} := V_\sigma$ 
3: while  $T_{\text{cur}} \neq T_{\text{prev}}$  do
4:    $T_{\text{prev}} := T_{\text{cur}}$ 
5:    $C := \text{SeqAttr}(G, T_{\text{prev}}, \sigma, \text{ParAlg})$ 
6:    $T_{\text{cur}} := T_{\text{prev}} \setminus \{v \in V_\sigma : N(v) \cap C = \emptyset\}$ 
7: end while
8: return  $C$ 
```

As before, TDA calls SeqAttr on progressively smaller sets of target vertices T_{cur} .

One easily sees that the set C output by $\text{TDA}(G, \sigma, \text{ParAlg})$ is the Sequential Attractor of some final collection $T \subseteq V_\sigma$ of target vertices. If σ follows the strategy given by the sequential attractor on C , the resulting play will either be winning for σ , as guaranteed by the conditions on ParAlg , or reach T infinitely often, and the largest priority will belong to σ and so the play will be winning for σ .

4.4. Correctness

We now outline the proof of [Theorem 4.4](#). We will prove three propositions from which [Theorem 4.4](#) will follow immediately. Note that, in the second statement below (completeness), V is the whole vertex set of the parity game.

Theorem 4.5. *Let $\text{ParAlg}(G, \sigma)$ be an algorithm that takes as input a parity game G and a player σ and returns a subset of the vertices of G . Assume ParAlg is nice with traps.*

- (1) (*Monotonicity*): If A is a $\bar{\sigma}$ -trap in G , then we have $\text{TDA}(G[A], \sigma, \text{ParAlg}) \subseteq \text{TDA}(G, \sigma, \text{ParAlg})$.
- (2) (*Completeness*): If V is good, then $\text{TDA}(G, \sigma, \text{ParAlg}) = V$.
- (3) (*Soundness*): $\text{TDA}(G, \sigma, \text{ParAlg})$ is good.

We will now build up the machinery to prove the above.

4.4.1. General lemmas

We begin with some general lemmas that will be useful; they are all reasonably simple to prove and we omit their proofs.

The first of these notes that SafeAttr is equal to Attr for λ large enough.

Lemma 4.6. $\text{SafeAttr}(G, \lambda, X, \sigma) = \text{Attr}(G, X, \sigma)$ if $\lambda > p(\max(V \setminus X))$.

The next lemma notes that the algorithms are stable when run on their own outputs. The second and third statements follow from the ones prior.

Lemma 4.7. *Let A be a $\bar{\sigma}$ trap.*

- (1) *If $S := \text{SafeAttr}(G, \lambda, X, \sigma) \subseteq A$, then*

$$S = \text{SafeAttr}(G, \lambda, S, \sigma) = \text{SafeAttr}(G[A], \lambda, S, \sigma).$$

- (2) *If $S := \text{GenAttr}(G, \lambda, X, \sigma, \text{ParAlg}) \subseteq A$, then*

$$S = \text{GenAttr}(G, \lambda, S, \sigma, \text{ParAlg}) = \text{GenAttr}(G[A], \lambda, S, \sigma, \text{ParAlg}).$$

- (3) *If $S := \text{SeqAttr}(G, \lambda, X, \sigma, \text{ParAlg}) \subseteq A$, then*

$$S = \text{SeqAttr}(G, \lambda, S, \sigma, \text{ParAlg}) = \text{SeqAttr}(G[A], \lambda, S, \sigma, \text{ParAlg}).$$

A similar statement holds for the TDA. Observe first that $\text{TDA}(G, \sigma, k)$ is a $\bar{\sigma}$ trap in G with maximum vertex of parity σ .

Lemma 4.8. *Take $S := \text{TDA}(G, \sigma, \text{ParAlg})$. Then*

$$S = \text{TDA}(G[S], \sigma, \text{ParAlg}).$$

As before, it is also true that $\text{TDA}(G, \sigma, \text{ParAlg}) = \text{TDA}(G[A], \sigma, \text{ParAlg})$ where A is any $\bar{\sigma}$ trap containing $\text{TDA}(G, \sigma, \text{ParAlg})$; this follows easily from the characterization of the TDA, but is more difficult to prove given only what we have established so far.

4.4.2. Monotonicity

We now prove monotonicity of SafeAttr for the inputs X, λ and sometimes for the graph G :

Lemma 4.9. *If $X_1 \subseteq X_2$, $\lambda_1 \leq \lambda_2$, and A is a $\bar{\sigma}$ trap with $X_1 \subseteq A$, then $\text{SafeAttr}(G[A], \lambda_1, X_1, \sigma) \subseteq \text{SafeAttr}(G, \lambda_2, X_2, \sigma)$.*

Proof. Take C_0, C_1, \dots to be the values of C_{cur} at the beginning of each “while” loop in the execution of $\text{SafeAttr}(G[A], \lambda_1, X_1, \sigma)$ (with the final value repeating) and take C'_0, C'_1, \dots similarly from the execution of $\text{SafeAttr}(G, \lambda_2, X_2, \sigma)$. Then $C_0 \subseteq C'_0$.

Note that, by definition of a trap, if $v \in A \cap V_\sigma$ then $N_{G[A]}(v) \subseteq N_G(v)$, and if $v \in A \cap V_{\bar{\sigma}}$ then $N_{G[A]}(v) = N_G(v)$.

If $C_i \subseteq C'_i$ then we have

$$\{v \in A \cap V_\sigma : p(v) < \lambda_1 \wedge N_{G[A]}(v) \cap C_i \neq \emptyset\} \subseteq \{v \in V_\sigma : p(v) < \lambda_2 \wedge N_G(v) \cap C'_i \neq \emptyset\},$$

$$\{v \in A \cap V_{\bar{\sigma}} : p(v) < \lambda_1 \wedge N_{G[A]}(v) \subseteq C_i\} \subseteq \{v \in V_{\bar{\sigma}} : p(v) < \lambda_2 \wedge N_G(v) \subseteq C'_i\}$$

and so $C_{i+1} \subseteq C'_{i+1}$, and by induction this holds for all i . \square

We now simultaneously address three monotonicity properties for GenAttr: monotonicity of the output with respect to the inputs X, λ and also sometimes with respect to the graph G .

The next lemma is a weak monotonicity property for GenAttr, saying that one iteration of the ‘WHILE’ loop in the algorithm will be contained in the GenAttr of the stronger inputs; combining this with the previous lemma will give monotonicity properties for GenAttr.

Lemma 4.10. Assume $X_1 \subseteq X_2$ and $\lambda_1 \leq \lambda_2$. Assume A is a $\bar{\sigma}$ trap in G and $X_1 \subseteq A$. Take

$$\begin{aligned} S^* &:= \text{SafeAttr}(G[A], \lambda_1, X_1, \sigma), \\ V^* &:= \text{Restrict}(G[A \setminus S^*], \lambda_1, \sigma), \\ T^* &:= \text{ParAlg}(G[V^*], \sigma). \end{aligned}$$

Then $S^* \cup T^* \subseteq \text{GenAttr}(G, \lambda_2, X_2, \sigma, \text{ParAlg})$.

Proof. Take C' to be the value of C_{cur} at the end of the execution of $\text{GenAttr}(G, \lambda_2, X_2, \sigma, \text{ParAlg})$. Taking:

$$\begin{aligned} S' &:= \text{SafeAttr}(G, \lambda_2, C', \sigma), \\ V' &:= \text{Restrict}(G[V \setminus S'], \lambda_2, \sigma), \\ T' &:= \text{ParAlg}(G[V'], \sigma) \end{aligned}$$

we have that $C' = S'$ and T' is empty.

We get that $S^* \subseteq S' = C'$ by monotonicity of the safe attractor, and so we need only show that $T^* \subseteq C'$. Assume, for sake of contradiction, that T^* is not a subset of C' , and so in particular $T^* \setminus S' \neq \emptyset$.

In the following, we refer to traps in induced subgraphs that are not necessarily subgames, i.e. they may have vertices without outgoing edges. The definition of trap remains unchanged.

We claim that $T^* \setminus S'$ is a $\bar{\sigma}$ trap in $G[V^*]$. Take $B := \{v \in A : p(v) \geq \lambda_1\}$. We know that T^* is a $\bar{\sigma}$ trap in $G[V^*]$. Then note that $V^* = (A \setminus S^*) \setminus \text{Attr}(G[A \setminus S^*], B, \bar{\sigma})$, and so we get that V^* is a $\bar{\sigma}$ trap in $G[A \setminus S^*]$ and so T^* is a $\bar{\sigma}$ trap in $G[A \setminus S^*]$. Since the edges of $G[A \setminus S']$ are a subset of the edges of $G[A \setminus S^*]$, we get that any $\bar{\sigma}$ vertex in $T^* \setminus S'$ has no edges leaving $T^* \setminus S'$ in the graph $G[A \setminus S']$. Given any σ vertex in $T^* \setminus S'$, since $S' = \text{SafeAttr}(G, \lambda_2, S', \sigma)$ and since $\forall v \in V^* p(v) < \lambda_1 \leq \lambda_2$, the σ vertex had no edges into S' in $G[A]$ (for otherwise it would be contained in S') and so the σ vertex must have some edge into $T^* \setminus S'$ and so we get that indeed $T^* \setminus S'$ is a $\bar{\sigma}$ trap in $G[A \setminus S']$, and so also in $G[V \setminus S']$ (since A is a $\bar{\sigma}$ -trap in V). We have $T^* \setminus S'$ is a $\bar{\sigma}$ -trap in $G[V \setminus S']$ with no vertices of priority at least λ ; any such structure must be a $\bar{\sigma}$ -trap in $G[V']$.

Because T^* has no vertices of priority at least λ_2 , we have $T^* \setminus S' = T^* \setminus \text{Attr}(G[T^*], S', \sigma)$, and so $T^* \setminus S'$ is a σ -trap in $G[T^*]$. Since we know ParAlg is nice with traps, we have that $\text{ParAlg}(G[T^*], \sigma) = T^*$. Therefore, again since ParAlg is nice with traps, $\text{ParAlg}(G[T^* \setminus S'], \sigma)$ is nonempty. Finally, $\text{ParAlg}(G[T^* \setminus S'], \sigma) \subseteq \text{ParAlg}(G[V'], \sigma)$, but this contradicts the assumption that $T' = \emptyset$. Therefore, we must have that $T^* \subseteq S'$. \square

Lemma 4.11. If $X_1 \subseteq X_2$ and $\lambda_1 \leq \lambda_2$ and A is a $\bar{\sigma}$ trap in G with $X_1 \subseteq A$, then $\text{GenAttr}(G[A], \lambda_1, X_1, \sigma, \text{ParAlg}) \subseteq \text{GenAttr}(G, \lambda_2, X_2, \sigma, \text{ParAlg})$.

Proof. Take C_0, C_1, \dots to be the values of C_{cur} at the beginning of each “while” loop in the execution of $\text{GenAttr}(G[A], \lambda_1, X_1, \sigma, \text{ParAlg})$ (with the final value repeating). Take

$$\begin{aligned} S_i &:= \text{SafeAttr}(G[A], \lambda_2, C_i, \sigma), \\ V_i &:= \text{Restrict}(G[A \setminus S_i], \lambda_2, \sigma), \\ T_i &:= \text{ParAlg}(G[V_i], \sigma). \end{aligned}$$

We will proceed by induction on i to show that $C_i \subseteq \text{GenAttr}(G, \lambda_2, X_2, \sigma, \text{ParAlg})$. Note this holds for C_0 since $C_0 = X_1 \subseteq X_2$. Then, for $i > 0$, we have $C_i = T_{i-1} \cup S_{i-1}$ and by the previous lemma and inductive hypothesis we get:

$$\begin{aligned} T_{i-1} \cup S_{i-1} &\subseteq \text{GenAttr}(G, \lambda_2, C_{i-1}, \sigma, \text{ParAlg}) \subseteq \text{GenAttr}(G, \lambda_2, \text{GenAttr}(G, \lambda_2, X_2, \sigma, \text{ParAlg}), \sigma, \text{ParAlg}) \\ &= \text{GenAttr}(G, \lambda_2, X_2, \sigma, \text{ParAlg}), \end{aligned}$$

completing the proof. \square

We will now proceed to show monotonicity of SeqAttr. While the original definition was slightly more natural, the following reformulation of SeqAttr will be more useful. We leave it to the reader to verify that the following reformulation of SeqAttr is equivalent to the original. It follows immediately from monotonicity and stability of GenAttr.

Lemma 4.12. *If P' is a finite collection of integers and $p(X \cap V_\sigma) \subseteq P'$ then, taking P to be the priorities in P' of parity σ , the following algorithm has the same output as SeqAttr.*

Algorithm 9 $\text{SeqAttr}_P(G = (V, E, p), X, \sigma, \text{ParAlg})$.

```

1:  $C := \emptyset$ 
2:  $Q := P$ 
3: while  $Q \neq \emptyset$  do
4:    $\lambda := \max(Q)$ 
5:    $Q := Q \setminus \{\lambda\}$ 
6:    $S := \{v \in X: p(v) = \lambda\}$ 
7:    $C := \text{GenAttr}(G, \lambda, C \cup S, \sigma, \text{ParAlg})$ 
8: end while
9: return  $C$ 

```

Intuitively, we simply run the GenAttr for every priority in P , which just adds redundancy by the assumption $p(X \cap V_\sigma) \subseteq P$: if ever in the original formulation of SeqAttr some call $\text{GenAttr}(G, \lambda, C, \sigma, \text{ParAlg})$ were made, then in the above version some call will be made with the same parameter λ .

We now show monotonicity properties for SeqAttr with respect to the input X and also sometimes with respect to the graph G :

Lemma 4.13. *If $X_1 \subseteq X_2$ and A is a $\bar{\sigma}$ -trap in G such that $X_1 \subseteq A$, then $\text{SeqAttr}(G[A], X_1, \sigma, \text{ParAlg}) \subseteq \text{SeqAttr}(G, X_2, \sigma, \text{ParAlg})$.*

Proof. Take $P = p(X_2 \cap V_\sigma)$. Take C_i, Q_i to be the values of C, Q respectively at the beginning of the i th iteration of the “WHILE” loop in the execution of $\text{SeqAttr}_P(G[A], X_1, \sigma, \text{ParAlg})$. Take

$$\begin{aligned} \lambda_i &= \max(Q_i), \\ S_i &= \{v \in X_1: p(v) = \lambda_i\}. \end{aligned}$$

Similarly take $C'_i, Q'_i, \lambda'_i, S'_i$ for the execution of $\text{SeqAttr}_P(G, X_2, \sigma, \text{ParAlg})$. Since $Q_0 = Q'_0$ and $Q_{i+1} = Q_i \setminus \max(Q_i)$ and $Q'_{i+1} = Q'_i \setminus \max(Q'_i)$ we get $Q_i = Q'_i$ and $\lambda_i = \lambda'_i$ for all i . Then $S_i \subseteq S'_i$ since $X_1 \subseteq X_2$. We now proceed by induction to show $C_i \subseteq C'_i$. We have $C_0 = C'_0 = \emptyset$ and

$$C_{i+1} = \text{GenAttr}(G[A], \lambda_i, S_i \cup C_i, \sigma, \text{ParAlg}) \subseteq \text{GenAttr}(G, \lambda'_i, S'_i \cup C'_i, \sigma, \text{ParAlg}) = C'_{i+1}. \quad \square$$

We now present the monotonicity theorem for TDA:

Theorem 4.14. *If A is a $\bar{\sigma}$ -trap in G , then we have $\text{TDA}(G[A], \sigma, \text{ParAlg}) \subseteq \text{TDA}(G, \sigma, \text{ParAlg})$.*

Proof. Let T_0, T_1, \dots be the values of T_{cur} at the beginning of the “WHILE” loop in the execution of $\text{TDA}(G[A], \sigma, \text{ParAlg})$. Take $C_i := \text{SeqAttr}(G[A], T_i, \sigma, \text{ParAlg})$. Similarly define T'_i, C'_i for $\text{TDA}(G, \sigma, \text{ParAlg})$. We proceed by induction to show $T_i \subseteq T'_i$. This holds for T_0, T'_0 since $A \cap V_\sigma \subseteq V_\sigma$. Then, by monotonicity of SeqAttr, we get $C_i \subseteq C'_i$. To obtain T_{i+1} and T'_{i+1} from T_i, T'_i , respectively, any vertex in T_i, T'_i without an edge into C_i, C'_i is removed. The edges of G are a superset of those of $G[A]$ and C'_i is a superset of C_i , so we get:

$$\begin{aligned} T_{i+1} &= T_i \setminus \{v \in V_\sigma \cap A: N_{G[A]}(v) \cap C_i = \emptyset\} = T_i \setminus \{v \in T_i: N_{G[A]}(v) \cap C_i = \emptyset\} \\ &\subseteq T'_i \setminus \{v \in T'_i: N_G(v) \cap C'_i = \emptyset\} = T'_i \setminus \{v \in V_\sigma: N_G(v) \cap C'_i = \emptyset\} = T'_{i+1}. \quad \square \end{aligned}$$

4.4.3. Completeness

Lemma 4.15. *If V is good for ParAlg with respect to σ and if $T \subseteq V$ and λ are such that $p(\max(V \setminus T)) < \lambda$, then taking $S := \text{GenAttr}(G, T, \lambda, \sigma, \text{ParAlg})$ we have $S = \text{Attr}(G, S, \sigma)$ and if $V \setminus S \neq \emptyset$ then $\max(V \setminus S) \subseteq V_\sigma$.*

Proof. We consider that by the terminating condition for the GenAttr algorithm, we must have

$$S = \text{SafeAttr}(G, S, \lambda, \sigma).$$

Note that $\text{SafeAttr}(G, S, \lambda, \sigma) = \text{Attr}(G, S, \sigma)$ (since $\lambda > p(\max(V \setminus T))$) and so we get

$$S = \text{Attr}(G, S, \sigma).$$

Since $p(\max(V \setminus S)) < \lambda$, we also get by the terminating condition for GenAttr that $\text{ParAlg}(V \setminus S, \sigma) = \emptyset$, but, by assumption, since $V \setminus S$ is a σ trap in G , either $V \setminus S = \emptyset$ (in which case we are done) or $\max(V \setminus S) \subseteq V_\sigma$. \square

Lemma 4.16. *If V is good for ParAlg with respect to σ , then $\text{SeqAttr}(G, V_\sigma, \sigma, \text{ParAlg}) = V$.*

Proof. Taking W_0, W_1, \dots to be the values of W at the beginning of each while-loop in the execution of $\text{SeqAttr}(G, V, \sigma, \text{ParAlg})$, take

$$S_i := \max(W_i),$$

$$C_i := \text{GenAttr}(G, p(S_i), C_{i-1} \cup S_i, \sigma, \text{ParAlg})$$

then we have by the previous lemma $\max(W_0) = \max(V)$. Then, by induction, if $\max(W_i) \in V_\sigma$, we have either $W_{i+1} = \emptyset$ or $\max(W_{i+1}) = \max(V \setminus C_i)$ and so the SeqAttr will not terminate until $V \subseteq C_i$. \square

Theorem 4.17. *If V is good for ParAlg with respect to σ , then $\text{TDA}(G, \sigma, \text{ParAlg}) = V$.*

Proof. The previous lemma immediately gives that the TDA will terminate after the first iteration of the “WHILE” loop and return V , since SeqAttr will return the whole set of vertices. \square

4.4.4. Soundness

Lemma 4.18. *If X is a σ -trap in G and if $X \cap Y = \emptyset$ then $X \cap \text{SafeAttr}(G, Y, \lambda, \sigma) = \emptyset$.*

Proof. Note that $\text{SafeAttr}(G, Y, \lambda, \sigma) \subseteq \text{Attr}(G, Y, \sigma)$ and $\text{Attr}(G, Y, \sigma) \cap X = \emptyset$. \square

Lemma 4.19. *If X is a σ -trap in G with largest vertex of priority $m < \lambda$ and with $\text{ParAlg}(G[X], \sigma) = \emptyset$, then if $X \cap Y = \emptyset$ we have $X \cap \text{GenAttr}(G, \lambda, Y, \sigma, \text{ParAlg}) = \emptyset$.*

Proof. Take C_0, C_1, \dots to be the value of C_{cur} at the beginning of each “WHILE” loop in the execution of $\text{GenAttr}(G, \lambda, Y, \sigma, \text{ParAlg})$. Take

$$S_i := \text{SafeAttr}(G, \lambda, C_i, \sigma),$$

$$V_i := \text{Restrict}(G[V \setminus S_i], \lambda, \sigma),$$

$$T_i := \text{ParAlg}(G[V_i], \sigma).$$

We proceed by induction on i to show $C_i \cap X = \emptyset$. This holds by assumption for $C_0 = Y$. If this holds for C_i , then $S_i \cap X = \emptyset$.

Note that, because all vertices in X have priority smaller than λ , $X \cap V_i = X \setminus \text{Attr}(G[X], X \setminus V_i, \bar{\sigma})$. In particular, we have that $X \cap V_i$ is a $\bar{\sigma}$ -trap in $G[X]$. Since ParAlg is nice with traps, this implies that $\text{ParAlg}(G[X \cap V_i], \sigma) = \emptyset$.

Since X is a σ -trap in G and $X \cap S_i = \emptyset$, we get that X is a σ -trap in $G[V \setminus S_i]$. By definition, V_i is the complement of a $\bar{\sigma}$ -attractor, so V_i is a $\bar{\sigma}$ -trap in $G[V \setminus S_i]$. Since X is a σ -trap and V_i is a $\bar{\sigma}$ -trap, we get that $X \cap V_i$ is a σ -trap in $G[V_i]$. Since T_i is a $\bar{\sigma}$ -trap in $G[V_i]$ and $X \cap V_i$ is a σ -trap in $G[V_i]$, we have that $X \cap V_i \cap T_i$ is a $\bar{\sigma}$ trap in $G[X \cap V_i]$. Therefore, since ParAlg is nice with traps, $\text{ParAlg}(G[X \cap V_i \cap T_i], \sigma) = \emptyset$. If $X \cap V_i \cap T_i$ were nonempty, then, since ParAlg is nice with traps and $X \cap V_i$ is a σ -trap in $G[V_i]$, we would have $\text{ParAlg}(G[X \cap V_i \cap T_i], \sigma) \neq \emptyset$, a contradiction. Therefore, we must have $X \cap V_i \cap T_i = X \cap T_i = \emptyset$.

Finally, since $C_{i+1} = S_i \cup T_i$, we have that $X \cap C_{i+1} = \emptyset$, as desired. \square

Lemma 4.20. *If X is a σ -trap with largest vertex of priority λ , λ has parity $\bar{\sigma}$, and $\text{ParAlg}(G[X], \sigma) = \emptyset$, then the largest vertices of X are not contained in $\text{SeqAttr}(G, V, \sigma, \text{ParAlg})$.*

Proof. Take $C_0 = \emptyset$ and W_0, W_1, \dots to be the value of W at the beginning of each “WHILE” loop in the execution of $\text{SeqAttr}(G, V, \sigma, \text{ParAlg})$. Take

$$S_i = \max(W_i),$$

$$C_i = \text{GenAttr}(G, p(S_i), C_{i-1} \cup S_i, \sigma, \text{ParAlg}).$$

If $p(S_i) > \lambda$ we have, by maximality of λ , that $X \cap S_i = \emptyset$, and so by induction that $X \cap (C_{i-1} \cup S_i) = \emptyset$. By the previous lemma we get $X \cap C_i = \emptyset$. If $p(S_i) < \lambda$, then we have $\max(X) \cap S_i = \emptyset$ and so by induction $\max(X) \cap (S_i \cup C_{i-1}) = \emptyset$. Therefore, no vertices of $\max(X)$ can be added by the call to GenAttr and so $\max(X) \cap C_i = \emptyset$. \square

Theorem 4.21. $\text{TDA}(G, \sigma, \text{ParAlg})$ returns a set that is good for σ with respect to ParAlg.

Proof. Take $S := \text{TDA}(G, \sigma, \text{ParAlg})$. Then $S = \text{TDA}(G[S], \sigma, \text{ParAlg})$. We've observed before that S is a $\bar{\sigma}$ trap whose largest vertex has priority of parity σ , so we may assume without loss of generality that $S = V$. By the previous lemma, there is no nonempty set X that is a σ -trap with largest vertex of priority $\bar{\sigma}$ so that $\text{ParAlg}(G[X], \sigma) = \emptyset$. \square

This completes the proof of [Theorem 4.4](#).

4.4.5. Second Trap-Depth Algorithm

We now define the second Trap-Depth Algorithm, $\text{TDA}(G, \sigma, k)$, and prove [Theorem 4.1](#). We will define $\text{TDA}(G, \sigma, k)$ by recursively applying $\text{TDA}(G, \sigma, \text{ParAlg})$. In order to do so, we will need to know that $\text{TDA}(G, \sigma, \text{ParAlg})$ is nice with traps whenever ParAlg is.

Lemma 4.22. If ParAlg is nice with traps, then $\text{TDA}(G, \sigma, \text{ParAlg})$ is nice with traps.

Proof. The same argument used in proving [Theorem 3.18](#) applies to show that, for any σ -trap Y , if $\text{TDA}(G, \sigma, \text{ParAlg}) \cap Y$ is nonempty, then $\text{TDA}(G[Y], \sigma, \text{ParAlg})$ is nonempty. The rest of the properties of being nice with traps follow from [Theorem 4.4](#). \square

We now define $\text{TDA}_k(G, \sigma) = \text{TDA}(G, \sigma, k)$ recursively. Define $\text{TDA}_0(G, \sigma)$ to be the algorithm that always returns the empty set. Note this is nice with traps. Given $\text{TDA}_k(G, \sigma)$, define $\text{TDA}_{k+1}(G, \sigma)$ by $\text{TDA}_{k+1}(G, \sigma) = \text{TDA}(G, \sigma, \text{TDA}_k)$. Inductively applying [Theorem 4.4](#) now proves [Theorem 4.1](#).

Note that we had previously defined $\text{TDA}_1(G, \sigma)$. Recalling that

$$\text{GenAttr}(G, \lambda, X, \sigma, \text{TDA}_0) = \text{SafeAttr}(G, \lambda, X, \sigma)$$

and that SafeAttr stabilizes its own output, it is easy to see that these two definitions match.

4.4.6. Runtime

Lemma 4.23. Let $T(n, m)$ be an upper bound on the runtime of $\text{ParAlg}(G, \sigma)$ for a graph G on n vertices and m edges. Then the runtime of $\text{TDA}(G, \sigma, \text{ParAlg})$ on a graph on n vertices and m edges is at most $O(mn^2) + n^2T(n, m)$.

Proof. Consider first the Safe Attractor Algorithm. Since each iteration of the “while” loop increases the size of C_{cur} or halts the algorithm, there will be at most $O(n)$ loops. If implemented carefully (in the same way that the regular Attractor is implemented) we may guarantee that each edge is only used a constant number of times and actually run the algorithm in $O(m + n) = O(m)$ time.

Next, consider the Generalized Safe Attractor Algorithm. Each iteration of the “while” loop increases the size of C_{cur} or halts the algorithm. On top of calling ParAlg, the algorithm does $O(m)$ work for each loop ($\text{Restrict}(G, \lambda, \sigma)$ can be computed in linear time). If the algorithm runs j “while” loops, it does work at most $(O(m) + T(n, m)) \times j$.

The Sequential Attractor Algorithm has C increasing every iteration or the algorithm halts. Note that each time a call to generalized attractor causes the generalized attractor to go through a “while” loop, a new vertex is added to C , so the total number of such loops done throughout the calls to generalized attractor is n , and so the total amount of work is at most $(O(m) + T(n, m)) \times n = O(mn) + nT(n, m)$.

In TDA we have T_{cur} decreasing on each iteration or the algorithm halts, and so there are at most n calls to SeqAttr, and on top of these only $O(m)$ work is done, and so we get $T(n, m) = O(mn^2) + n^2T(n, m)$. \square

Lemma 4.24. Let $T(n, m, k)$ denote the runtime of $\text{TDA}(G, \sigma, k)$ for a graph G on n vertices and m edges. Then $T(n, m, 0) = O(1)$ and for $k > 0$ we have $T(n, m, k) = O(mn^{2k-1})$.

Proof. We have $T(n, m, 0) = O(1)$ since this algorithm always returns the empty set.

The same optimizations used in the computation of the Attractor and the Safe Attractor may be used to get a runtime of $O(m)$ in the case $k = 1$ for the sequential attractor, that is for SeqAttr_1 . In TDA we have T_{cur} decreasing on each iteration or the algorithm halts, and so there are at most n calls to SeqAttr_1 , and on top of these only $O(m)$ work is done, and so we get $T(n, m, 1) = O(mn)$.

For $k \geq 1$ by the previous lemma we have $T(n, m, k + 1) \leq O(mn^2) + n^2T(n, m)$. This recurrence solves to $T(n, m, k) = O(mn^{2k-1})$, as desired. \square

5. Summary and critical remarks

The theorems of the previous section show the promised characterization of TDA ([Theorem 4.1](#)):

$\text{TDA}(G, \sigma, k)$ returns the largest (possibly empty) set starting with which σ can guarantee a win in at most k moves in the trap-depth game on G .

We have introduced Trap-Depth games (where the moves consist of choosing subsets of the graph rather than vertices/edges) and shown their close relationship with Muller games. We have defined the trap-depth parameter and given algorithms for parity games for finding subsets of the winning regions whose runtime is bounded by an exponential in this trap-depth. Writing $d := |p(V)|$, since the trap-depth of a parity game is at most $\lceil \frac{d}{2} \rceil$, the algorithm runs in time $O(mn^d)$. If one is only interested in the class of graphs with a bounded number of priorities, there are other options. The classical algorithm of Zielonka also runs in time $O(mn^d)$ (see [7]), but there are better algorithms: Jurdziński's [10] algorithm achieves $O(dm(\frac{n}{\lfloor \frac{d}{2} \rfloor})^{\lfloor \frac{d}{2} \rfloor})$, and the subexponential algorithm of [11] as well as a polynomial improvement thereof by [16] achieve $n^{O(\sqrt{n})}$ and $mn^{\frac{d}{2} + O(1)}$. Of course, the class of graphs of bounded trap depth is much more general than the class of graphs with a bounded number of priorities.

By Lemma 2.4, finding any nonempty subset of the winning region allows us to remove part of the graph to get a smaller parity game that needs to be solved; thus, for example, Parity games in which every subgame has bounded trap depth (such as those with a bounded number of priorities) may be completely solved in polynomial time, a generalization of the result that parity games with a bounded number of priorities may be solved in polynomial time.

Parity games are just one encoding of a class of Muller games. One may ask if there are others for which the characterization of Muller games we present is algorithmically useful. One possible encoding is called *Explicit Muller games*, where an enumeration of the sets winning for Red, i.e. of the set \mathcal{R} , is explicitly given as input. There is a known polynomial time algorithm for solving explicit Muller games [6], but we may hope to obtain another algorithm using the characterization. If one could efficiently answer the following question, such an algorithm exists (note in the following question (V, E, V_{red}) are given explicitly):

Problem 5.1. Given a Muller game G and an explicit list $S_1, \dots, S_k \subseteq V$, is there some polynomial time algorithm that determines if every Red-trap H contains one of the S_i as a Blue-subtrap?

To see that the above would allow us to solve the problem, let an explicit Muller game $(V, E, V_{\text{red}}, \mathcal{R})$ be given. We will first prune \mathcal{R} by removing any sets $R \in \mathcal{R}$ in which some vertex has no outgoing edges in $G[R]$ (these have no impact on the game). To determine if Red has a nonempty winning region, we will find the collection W of sets in \mathcal{R} from which Red will win the trap-depth game in which Blue goes first.

We will iteratively update \mathcal{R} and W . Choose any minimal (under inclusion) set $R \in \mathcal{R}$. For each such set R we determine if $G[R]$ contains any Red-traps that do not contain as a Blue-trap any set in $W \cup \{R\}$. If $G[R]$ has no such Red-traps, then we add R to W . In either case, we remove R from \mathcal{R} and iterate.

It is easy to argue that if in the trap-depth game the set of vertices is X and it is Red's turn to move, then a Blue-trap Y in $G[X]$ is winning for Red if and only if H is in W . To determine if Red has a non-empty winning region, we need only check if one of the sets in W is a Blue-trap in G .

Acknowledgements

This work was partially supported by NSF grant DMS-0648208 at the Cornell REU, which are both gratefully acknowledged. Andrey Grinshpun is partially supported by the NPSC. Andrei Tarfulea is partially supported by the NSF GRFP. We warmly thank Alex Kruckman, James Worthington and Ben Zax for many stimulating discussions on an early part of this work, as well as Damian Niwinski for his comments. We also thank the anonymous referee, without whose comments reading this paper would be much less pleasant.

References

- [1] D. Berwanger, A. Dawar, P. Hunter, S. Kreutzer, DAG-width and parity games, in: STACS 2006, in: Lect. Notes Comput. Sci., vol. 3848, 2006, pp. 524–536.
- [2] D. Berwanger, E. Grädel, Fixed-point logics and solitaire games, Theory Comput. Syst. 37 (2004) 675–694.
- [3] H. Björklund, S. Sandberg, S. Vorobyov, A discrete subexponential time algorithm for parity games, in: STACS 2003, in: Lect. Notes Comput. Sci., vol. 2607, 2003, pp. 663–674.
- [4] E. Emerson, C. Jutla, Tree automata, μ -calculus, and determinacy, in: Proceedings of the 32nd Annual Symposium on Foundations of Computer Science, IEEE, 1991, pp. 368–377.
- [5] E. Emerson, C. Jutla, A. Sistla, On model checking for fragments of μ -calculus, in: Computer Aided Verification, STACS 2006, in: Lect. Notes Comput. Sci., vol. 497, 1993, pp. 385–396.
- [6] F. Horn, Explicit Muller games are PTIME, in: Annual Conference on Foundations of Software Technology and Theoretical Computer Science, 2008.
- [7] E. Grädel, W. Thomas, T. Wilke, Automata, Logics, and Infinite Games, Springer, 2002.
- [8] P. Hunter, Complexity and infinite games on finite graphs, Ph.D. thesis, University of Cambridge, 2007.
- [9] M. Jurdziński, Deciding the winner in parity games is in $UP \cap co\text{-}UP$, Inf. Process. Lett. 68 (1998) 119–124.
- [10] M. Jurdziński, Small progress measures for solving parity games, in: STACS 2000, in: Lect. Notes Comput. Sci., vol. 1770, 2000, pp. 290–301.
- [11] M. Jurdziński, M. Paterson, U. Zwick, A deterministic subexponential time algorithm for solving parity games, in: Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms, Symposium on Discrete Mathematics, 2006, pp. 117–123.
- [12] D. Martin, Borel determinacy, Ann. Math. (2) 102 (2) (1975) 363–371.

- [13] R. McNaughton, Infinite games played on finite graphs, *Ann. Pure Appl. Log.* 65 (2) (1993) 149–184.
- [14] J. Obdržálek, Clique-width and parity games, in: *Computer Science Logic*, in: *Lect. Notes Comput. Sci.*, vol. 4646, 2007, pp. 54–68.
- [15] J. Obdržálek, Fast mu-calculus model checking when tree-width is bounded, in: *Computer Aided Verification*, in: *Lect. Notes Comput. Sci.*, vol. 2825, 2003, pp. 80–92.
- [16] S. Schewe, Solving parity games in big steps, in: *Foundations of Software Technology and Theoretical Computer Science*, in: *Lect. Notes Comput. Sci.*, vol. 4855, 2007, pp. 449–460.
- [17] W. Thomas, Facets of synthesis: revisiting Church's problem, in: *FOSSACS*, 2009, pp. 1–14.
- [18] W. Zielonka, Infinite games on finitely coloured graphs with applications to automata on infinite trees, *Theor. Comput. Sci.* 200 (1998) 135–183.

On CTL* with Graded Path Modalities*

Benjamin Aminof¹, Aniello Murano², and Sasha Rubin²

¹ Technische Universität Wien, Austria

² Università degli Studi di Napoli “Federico II”, Italy

Abstract. Graded path modalities count the number of paths satisfying a property, and generalize the existential (E) and universal (A) path modalities of CTL*. The resulting logic is denoted GCTL*, and is a very powerful logic since (as we show) it is equivalent, over trees, to monadic path logic. We settle the complexity of the satisfiability problem of GCTL*, i.e., 2EXPTIME-COMPLETE, and the complexity of the model checking problem of GCTL*, i.e., PSPACE-COMPLETE. The lower bounds already hold for CTL*, and so we supply the upper bounds. The significance of this work is two-fold: GCTL* is much more expressive than CTL* as it adds to it a form of quantitative reasoning, and this is done at no extra cost in computational complexity.

1 Introduction

Quantitative Verification and Graded Modalities. Temporal logics are the cornerstone of the field of formal verification. In recent years, much attention has been given to extending these by quantitative measures of function and robustness, e.g., [18]. Unfortunately, these extensions often require one to reason about weighted automata for which much is undecidable [1, 2, 10]. One way to extend classical temporal logics at a lower cost is by counting quantifiers, known as graded modalities. Graded world modalities were introduced in formal verification as a useful extension of the standard existential and universal quantifiers in branching-time modal logics [7, 16, 19, 23]. These modalities allow one to express properties such as “there exist at least n successors satisfying a formula” or “all but n successors satisfy a formula”. A prominent example is the extension of μ -calculus called $G\mu$ -calculus [7, 19].

Despite its high expressive power, the μ -calculus (which extends modal logic by least and greatest fixpoint operators) is a low-level logic, making it “unfriendly” for users, who usually find it very hard to understand, let alone write, formulas involving even very modest nesting of fixed points. In contrast, CTL and CTL* are much more intuitive and user-friendly. An extension of CTL with graded *path modalities* called GCTL was defined in [5, 6]. Although there are

* Benjamin Aminof is supported by the Austrian National Research Network S11403-N23 (RiSE) of the Austrian Science Fund (FWF) and by the Vienna Science and Technology Fund (WWTF) through grant ICT12-059. Aniello Murano is partially supported by the FP7 EU project 600958-SHERPA. Sasha Rubin is a Marie Curie fellow of the Istituto Nazionale di Alta Matematica.

several positive results about GCTL this logic suffers from similar limitations as CTL, i.e., it cannot nest successive temporal operators and so cannot express fairness constraints. This dramatically limits the usefulness of GCTL and so we turn instead to GCTL* in which one can naturally and comprehensibly express complex properties of systems. Although the syntax and semantics of GCTL* were defined and justified in [6], only a rudimentary study of it was made. In particular, the complexity of the satisfiability and model checking problem for this logic was never established, and remained open since its introduction in 2009. Instead, research has focused on the much simpler fragment of GCTL.

Our results. We establish the exact complexity of the satisfiability and model checking problems for GCTL* to be 2EXPTIME-COMPLETE and PSPACE-COMPLETE, respectively. Thus, in both cases, the problems for GCTL* are not harder than for CTL*. This is very good news indeed since, as we also show, GCTL* is expressively equivalent, over trees, to monadic path logic, and is thus a powerful, yet relatively friendly logic. Along the way, we prove that GCTL* has the bounded-degree tree-model property, i.e., a satisfiable formula is satisfied in a tree whose branching degree is at most exponential in the size of the formula.

The importance of our results. We obtain that GCTL* has the following desirable combination of attributes:

a) **GCTL* can naturally express properties of paths as well as count them.** For example, the formula $E^{\geq 2}G(request \rightarrow (request \cup granted))$ says: “there are at least two ways to schedule the computation such that every request is eventually granted”. This cannot be expressed in CTL* nor in GCTL.

The naive semantics for $E^{\geq n}\psi$ which states that “there are at least n different paths satisfying ψ ” while at first glance may seem natural and desirable, when examined more carefully turns out to be undesirable, and less informative. For example, consider a faulty program in which requests are sometimes not granted. In GCTL* (unlike the naive counting) the formula $E^{\geq 2}[F(request \wedge \neg F granted)]$ requires at least two *incomparable* sequences of operations, each causing this faulty behaviour. Hence, it indicates whether the faulty behaviour is the result of multiple underlying problems, and is not confused by multiple paths that are extensions of a single faulty prefix. Furthermore, the naive counting very quickly leads to unnatural interpretations, as convincingly argued in [6].

This ability to easily count paths is a natural fit in various application domains. For example, in databases there is a close relationship between model-checking CTL* and XML navigation (see [4]). The logic GCTL* allows one to express quantitative requirements such as ”client has at least 5 items in last-month orders”. More generally, graded operators are common in description logics, which are prominently used for formal reasoning in AI (e.g., knowledge querying, planning with redundancies).

b) **GCTL* is extremely expressive.** Not only does GCTL* extend CTL* (and thus, unlike CTL, it can reason about fairness), we prove that it is expressively equivalent, over trees, to Monadic Path Logic (MPL) which is Monadic Second-Order Logic (MSOL) interpreted over trees but with set quantification restricted to branches.

c) **GCTL*** has relatively low complexity of satisfiability. Unfortunately, the complexity of satisfiability of MPL is non-elementary (this is already true for FOL). In sharp contrast, we prove that the complexity of satisfiability of GCTL* is 2EXPTIME, and thus is no harder than for CTL*.

Technical Contributions. The upper bounds are obtained by exploiting an automata-theoretic approach for branching-time logics, combined with game theoretic reasoning at a crucial point. The automata-theoretic approach is suitable because GCTL* turns out to have the tree-model property. It is very hard to see how other techniques for deciding questions in logic (e.g. effective quantifier elimination, tableaux, composition) can be used to achieve optimal complexity results for GCTL*. Our proof is not just an easy adaptation of the classical decision procedure. We relate GCTL* to a new model of automata, i.e., *Graded Hesitant Tree Automata* (GHTA). These automata work on finitely-branching trees (not just k -ary trees) and their transition relations can count up to a given number (usual alternating automata only count up to 1).

Related Work. Counting modalities were first introduced by Fine [16] under the name *graded world modalities*. A systematic treatment of the complexity of various graded modal logics followed [9, 14, 21, 23, 24]. The extension of μ -calculus by graded world modalities was investigated in [7, 19]. Although these articles introduce automata that can count, our GHTA are more complicated since they have to deal with graded path modalities and not just graded world modalities. The extension of CTL* by the ability to say “there exist at least n successors satisfying ψ ”, called counting-CTL*, was defined in [22], and its connection with Monadic Path Logic studied using the composition method. It is unclear if that method, although elegant, can yield the complexity bounds we achieve (even for counting-CTL*). As shown in [6], $G\mu$ -calculus cannot succinctly reason about paths, or even grandchildren of a given node (the same goes for counting-CTL*). The first work to deal with graded path modalities is [5] that introduced GCTL, the extension of CTL by these modalities. Graded path modalities over CTL were also studied in [15], using a different semantics than GCTL which is tailored for extending CTL, and it is unclear how one can extend their work to CTL*.

2 The GCTL* temporal logic

Let \mathbb{N} denote the positive integers, and $[d] = \{1, 2, \dots, d\}$ for $d \in \mathbb{N}$. An LTS (Labeled Transition System/Kripke structure) is a tuple $S = \langle \Sigma, S, E, \lambda \rangle$, where Σ is a set of *labels*, S is a countable set of *states*, $E \subseteq S \times S$ is the *transition relation*, and $\lambda : S \mapsto \Sigma$ is the *labeling function*. Typically, $\Sigma = 2^{\text{AP}}$ where AP is a finite set of *atomic propositions*. The *degree* of a state s is the cardinality of the set $\{t \in S : (s, t) \in E\}$ of its successors. We assume that E is total, i.e., that every state has a successor. A path in S is a finite or infinite sequence $\pi_0\pi_1\dots \in (S^*) \cup (S^\omega)$ such that $(\pi_{i-1}, \pi_i) \in E$ for all $1 \leq i < |\pi|$ ($|\pi|$ is the *length* of π). The set of (finite and infinite) paths in S is written $\text{pth}(S)$, and the set of (finite and infinite) paths in S that start in a given state $q \in S$ is written $\text{pth}(S, q)$. Let \preceq be the prefix ordering on paths. If $\pi \preceq \pi'$ say that π' is an *extension* of

π . For a set of paths X , denote by $\min(X)$ the minimal elements of X according to \preceq . A Σ -labeled tree T is a pair $\langle T, V \rangle$ where $T \subseteq \mathbb{N}^*$ is a \prec -downward closed set of strings over \mathbb{N} , and $V : T \rightarrow \Sigma$ is a labeling. We implicitly view a tree $T = \langle T, V \rangle$ as the LTS $\langle \Sigma, T, E, V \rangle$ where $(t, s) \in E$ iff s is a son of t . If every node of a tree T has a finite degree then T is *finitely branching*. If every node has at most degree $k \in \mathbb{N}$, then T is *boundedly branching* or *has branching degree k* .

2.1 Syntax and Semantics of GCTL*

GCTL* extends CTL* by *graded path quantifiers* of the form $E^{\geq g}$. We follow the definition of GCTL* from [6], but give a slightly simpler syntax. We assume that the reader is familiar with the logics CTL*, LTL, and CTL (see [20, 25]).

The semantics of GCTL* is defined for an LTS S . The GCTL* formula $E^{\geq g}\psi$, for GCTL* path formula ψ , can be read as “*there exist at least g (minimal ψ -conservative) paths*”. Minimality was defined above, and so we now say, informally, what it means for a path to be ψ -conservative. An infinite path of S is ψ -conservative if it satisfies ψ , and a finite path of S is ψ -conservative if all its (finite and infinite) extensions in S satisfy ψ . Note that this notion uses a semantics of GCTL* over finite paths, and thus the semantics of GCTL* needs to be defined for finite paths (as well as infinite paths). As in [6], we use the weak-version of semantics of temporal operators for finite paths (defined in [11]). Intuitively, temporal operators are interpreted pessimistically (with respect to possible extensions of the path), e.g., $(S, \pi) \models X\psi$ iff $|\pi| \geq 2$ and $(S, \pi_{\geq 1}) \models \psi$.

Syntax of GCTL*. Fix a set of atoms AP. The GCTL* *state* (φ) and *path* (ψ) formulas are built inductively from AP using the following grammar:

$$\varphi ::= p \mid \neg\varphi \mid \varphi \vee \varphi \mid E^{\geq g}\psi \text{ and } \psi ::= \varphi \mid \neg\psi \mid \psi \vee \psi \mid X\psi \mid \psi U \psi \mid \psi R \psi.$$

In the first part, p varies over AP and g varies over \mathbb{N} (and thus, technically, there are infinitely many rules in this grammar). As usual, X , U and R are called *temporal operators* and $E^{\geq g}$ (for $g \in \mathbb{N}$) are called *path modalities* (also called *path quantifiers*). We write $F\varphi$ instead of $\text{true}U\varphi$, and $G\varphi$ instead of $\text{false}R\varphi$. The class of GCTL* *formulas* is the set of state formulas generated by the above grammar. The simpler class of *Graded CTL formulas* (GCTL) is obtained by requiring each temporal operator to be immediately preceded by a path quantifier. The logic LTL is the class of path formulas in which no path quantifier appears. The *degree* of the quantifier $E^{\geq g}$ is the number g . The *degree* $\deg(\varphi)$, of a state formula φ , is the maximum of the degrees of the quantifiers appearing in φ . The *length* $|\varphi|$, of a formula φ , is defined inductively on the structure of φ as usual, and using $|E^{\geq g}\psi|$ equal to $g + 1 + |\psi|$ (i.e., g is coded in unary).

Semantics of GCTL*. Given an LTS S and a state $s \in S$, the definition of $(S, s) \models \varphi$ is done inductively on the structure of φ , exactly as for CTL*, with the only change concerning the new path quantifier $E^{\geq g}$. For $\varphi = E^{\geq g}\psi$, where ψ is a GCTL* path formula, let $(S, s) \models \varphi$ iff the cardinality of the set $\min(Con(S, s, \psi))$ is at least g , where $Con(S, s, \psi) := \{\pi \in pth(S, s) \mid \forall \pi' \in pth(S, s) : \pi \preceq \pi' \text{ implies } (S, \pi') \models \psi\}$. The paths in $Con(S, s, \psi)$ are called ψ -*conservative* (in S starting at s), and paths in $\min(Con(S, s, \psi))$ are called *minimal ψ -conservative*. It is not hard to see that, for total LTSSs, the classic

logic CTL* coincides with the fragment of GCTL* in which the degree g of all quantifiers $E^{\geq g}$ is 1.

If ψ is an LTL formula, we may write $\pi \models \psi$ instead of $(S, \pi) \models \psi$. This is justifiable since the truth of ψ depends only on the path π independently of the rest of S . Two state formulas ϕ, ϕ' are *equivalent* if for all S and $s \in S$, we have $(S, s) \models \phi$ iff $(S, s) \models \phi'$. Two path formulas ψ, ψ' are *equivalent* if for all S and $\pi \in \text{pth}(S)$, we have that $(S, \pi) \models \psi$ if and only if $(S, \pi) \models \psi'$. An LTS S with a designated state $q \in S$ is a *model* of a GCTL* formula φ , sometimes denoted $S \models \varphi$, if $(S, q) \models \varphi$. For a labeled tree T , the designated node is by default the root, and thus, $T \models \varphi$ means that $(T, \epsilon) \models \varphi$ (recall that ϵ designates the root of T). A GCTL* formula φ is *satisfiable* iff it has a model.

Example 1. We unpack the meaning of the GCTL* formula from the introduction $E^{\geq 2}[\mathsf{F}(\mathsf{request} \wedge \neg \mathsf{Granted})]$. Let ψ denote the path formula $\mathsf{F}(\mathsf{request} \wedge \neg \mathsf{Granted})$. First, a finite or infinite path π satisfies ψ if at some point t the atom *request* holds, and at no later point on π does the atom *granted* hold. A finite π is ψ -conservative if and only if it satisfies ψ and the atom *granted* does not hold in any node of the subtree rooted at the end of π ; and an infinite path is ψ -conservative if and only if it satisfies ψ . Thus, $E^{\geq 2}\psi$ holds if and only if there exist two possibly finite paths, say π^1 and π^2 , neither one a prefix of the other, both satisfying ψ (i.e., π^i has a request that is never granted on π^i), and such that if π^i is finite then that path has a request that is not granted in any possible extension of π^i .

2.2 Important Properties of GCTL*

Like CTL* (see [20]), one can think of a GCTL* path formula ψ over atoms AP as an LTL formula Ψ over atoms which themselves are GCTL* state formulas, as follows. A formula φ is a *state sub-formula* of ψ if i) φ is a state formula, and ii) φ is a sub-formula of ψ . A formula φ is a *maximal state sub-formula* of ψ if φ is a state sub-formula of ψ , and φ is not a proper sub-formula of any other state sub-formula of ψ . Let $\underline{\max}(\psi) = \{\varphi \mid \varphi \text{ is a maximal state sub-formula of } \psi\}$, and let $\overline{\max}(\psi) = \bigcup_{\varphi \in \underline{\max}(\psi)} \{\varphi, \neg\varphi\}$ be the set of all maximal state sub-formulas of ψ and their negations. Every GCTL* path formula ψ can be viewed as the formula Ψ whose atoms are elements of $\underline{\max}(\psi)$. Note that Ψ is an LTL formula. For example, for $\psi = ((Xp) U (E^{\geq 2}Xq)) \vee p$, the state sub-formulas are $\{p, q, E^{\geq 2}Xq\}$, and $\underline{\max}(\psi) = \{p, E^{\geq 2}Xq\}$, and thus Ψ is the LTL formula $(Xp) U \underline{E^{\geq 2}Xq} \vee \underline{p}$ over the atoms $\{p, E^{\geq 2}Xq\}$ (here we underline sub-formulas that are treated as atoms). Given an LTS $S = \langle 2^{AP}, S, E, \lambda \rangle$ and a GCTL* path formula ψ , we define the *relabelling* of the LTS S by the values of the formulas in $\underline{\max}(\psi)$ as $S_\psi = \langle \underline{\max}(\psi), S, E, L \rangle$ where $L(s)$ is the union of $\lambda(s)$ and the set of $\varphi \in \underline{\max}(\psi)$ such that $(S, s) \models \varphi$.

Lemma 1. *For every GCTL* path formula ψ over AP there is an LTL formula Ψ over $\underline{\max}(\psi)$ such that for all S and all paths π in S : $(S, \pi) \models \psi$ iff $(S_\psi, \pi) \models \Psi$.*

It is not hard to see that GCTL* is not invariant under bisimulation (cf. [6]), and that it is invariant under unwinding (cf. [6]). The next theorem shows that GCTL* is a powerful logic. Indeed, it is equivalent, over trees, to Monadic Path Logic (MPL) which is MSO with quantification restricted to branches. Note that MPL is only defined over trees, while GCTL* (like CTL*) is defined over arbitrary LTS. This is the reason we compare their expressiveness over trees.

Theorem 1. *GCTL* is equivalent, over trees, to Monadic Path Logic.*

3 Graded Hesitant Tree Automata

In this section we define a new kind of automaton called Graded Hesitant Tree Automata. We also make use of the classical non-deterministic finite word automata (NFW) and non-deterministic Büchi word automata (NBW) (see [25]), alternating parity tree automata (APTA) (see [12]), and alternating hesitant tree automata (AHTA) (see [20]). We write $\langle \Sigma, Q, q_0, \delta, G \rangle$ for NBWs and $\langle \Sigma, Q, q_0, \delta, F \rangle$ for NFWs where Σ is the input alphabet, Q is the set of states, q_0 is the initial state, $\delta \subseteq Q \times \Sigma \times Q$ is the transition relation, $G \subseteq Q$ is the set of accepting states and $F \subseteq Q$ the set of final states. For a set X , let $B^+(X)$ be the set of positive Boolean formulas over X , including the constants **true** and **false**. A set $Y \subseteq X$ satisfies a formula $\theta \in B^+(X)$, written $Y \models \theta$, if assigning **true** to elements in Y and **false** to elements in $X \setminus Y$ makes θ true. Graded hesitant tree automata (GHTA) generalise AHTA³: a) they can work on finitely-branching trees (not just k -ary branching trees), and b) their transition relation allows the automaton to send multiple copies into the successors of the current node in a much more flexible way. Below we formally define AHTA and GHTA.

Definition of AHTA An *Alternating Hesitant Tree Automaton* (AHTA) is a tuple $A = \langle \Sigma, D, Q, q_0, \delta, \langle G, B \rangle, \langle \text{part}, \text{type}, \preceq \rangle \rangle$ where Σ is a non-empty finite set of *input letters*; $D \subset \mathbb{N}$ is a finite non-empty set of *directions*, Q is the non-empty finite set of *states*, $q_0 \in Q$ is the *initial state*; the pair $\langle G, B \rangle \in 2^Q \times 2^Q$ is the *acceptance condition*⁴ (we sometimes call the states in G *good states* and the states in B *bad states*); $\delta : Q \times \Sigma \rightarrow B^+(D \times Q)$ is the *alternating transition function*; $\text{part} \subset 2^Q$ is a partition of Q , $\text{type} : \text{part} \rightarrow \{\text{trans}, \text{exist}, \text{univ}\}$ is a function assigning the label *transient*, *existential* or *universal* to each element

³ Strictly speaking, GHTA generalise the symmetric variant of AHTA. That is, for every language accepted by an AHTA and that is closed under the operation of permuting siblings, there is a GHTA that accepts the same language.

⁴ The combination of a Büchi and a co-Büchi condition that hesitant automata use can be thought of as a special case of the parity condition with 3 colors. Thus, we could have defined Graded Parity Tree Automata instead (using the parity condition, our automata strictly generalise the ones in [5, 19]) However, we do not need the full power of the parity condition, and in order to achieve optimal complexity for model checking of GCTL* we need to be able to decide membership of our automata in a space efficient way, which cannot be done with the parity acceptance condition.

of the partition, and $\preceq \subset 2^Q \times 2^Q$ is a partial order on part . Moreover, the transition function δ is required to satisfy the following *hesitancy condition*: for every $\mathbb{Q} \in \text{part}$, every $q \in \mathbb{Q}$, and every $\sigma \in \Sigma$: (i) for every $\mathbb{Q}' \in \text{part}$ and $q' \in \mathbb{Q}'$, if q' occurs in $\delta(q, \sigma)$ then $\mathbb{Q}' \preceq \mathbb{Q}$; (ii) if $\text{type}(\mathbb{Q}) \in \text{trans}$ then no state of \mathbb{Q} occurs in the formula $\delta(q, \sigma)$; (iii) if $\text{type}(\mathbb{Q}) \in \text{exist}$ (resp., $\text{type}(\mathbb{Q}) \in \text{univ}$) then there is at most one element of \mathbb{Q} in each disjunct of the DNF (resp., conjunct of CNF) of $\delta(q, \sigma)$.

An *input tree* (for AHTA) is a Σ -labeled tree $\mathsf{T} = \langle T, V \rangle$ with $T \subseteq D^*$. Since D is finite, such trees have fixed finite branching degree. A *run* (or *run tree*) of an alternating tree automaton \mathbf{A} on input tree $\mathsf{T} = \langle T, V \rangle$ is a $(T \times Q)$ -labeled tree $\langle T_r, r \rangle$, such that (a) $r(\varepsilon) = (\varepsilon, q_0)$ and (b) for all $y \in T_r$, with $r(y) = (x, q)$, there exists a *minimal* set $S \subseteq D \times Q$, such that $S \models \delta(q, V(x))$, and for every $(d, q') \in S$, it is the case that $x \cdot d$ is a son of x , and there exists a son y' of y , such that $r(y') = (x \cdot d, q')$.

Note that if $\delta(q, V(x)) = \text{true}$ then $S = \emptyset$ and the node y has no children; and if there is no S as required (for example if x does not have the required sons) then there is no run-tree with $r(y) = (x, q)$. Observe that disjunctions in the transition relation are resolved into different run trees, while conjunctions give rise to different sons of a node in a run tree. If v is a node of the run tree, and $r(v) = (u, q)$, call u the *location associated with* v , denoted $\text{loc}(v)$, and call q the *state associated with* v , denoted $\text{state}(v)$.

We now discuss the acceptance condition. Fix a run tree $\langle T_r, r \rangle$ and an infinite path π in it. Say that the path *visits* a state q at time i if $\text{state}(\pi_i) = q$. The hesitancy restriction (i) guarantees that the path π eventually gets trapped and visits only states in some element of the partition, i.e., there exists $\mathbb{Q} \in \text{part}$ such that from a certain time i on, $\text{state}(\pi_j) \in \mathbb{Q}$ for all $j \geq i$. The condition (ii) ensures that this set is either existential or universal, i.e., $\text{type}(\mathbb{Q}) \in \{\text{exist}, \text{univ}\}$. Thus, we say that the path π *gets trapped in an existential set* if $\text{type}(\mathbb{Q}) = \text{exist}$, and otherwise we say that it *gets trapped in a universal set*. We can now define what it means for a path in a run tree to be *accepting*. A path that gets trapped in an existential set is *accepting* iff it visits some state of G infinitely often, and a path that gets trapped in a universal set is *accepting* iff it visits every state of B finitely often. A run $\langle T_r, r \rangle$ of an AHTA is *accepting* iff all its infinite paths are accepting. An automaton \mathbf{A} accepts an input tree $\langle T, V \rangle$ iff there is an accepting run of \mathbf{A} on $\langle T, V \rangle$. The *language* of \mathbf{A} , denoted $\mathcal{L}(\mathbf{A})$, is the set of Σ -labeled D -trees accepted by \mathbf{A} . We say that \mathbf{A} is nonempty iff $\mathcal{L}(\mathbf{A}) \neq \emptyset$.

The *membership problem* of AHTA is the following decision problem: given an AHTA \mathbf{A} with direction set D , and a finite LTS S in which the degree of each node is at most $|D|$, decide whether or not \mathbf{A} accepts S . The *depth* of the AHTA is the size of the longest chain in \prec . The *size* $\|\delta\|$ of the transition function is the sum of the lengths of the formulas it contains. The *size* $\|\mathbf{A}\|$ of the AHTA is $|D| + |Q| + \|\delta\|$. The partition, partial order and type function are not counted in the size of the automaton. The following is implicit in [20]:

Theorem 2. *The membership problem for AHTA can be solved in $O(\partial \log^2(|S| \cdot \|\mathbf{A}\|))$ space where ∂ is the depth of \mathbf{A} and S is the state set of S .*

Definition of GHTA We now introduce *Graded Hesitant Tree Automata* (GHTA). These can run on finitely-branching trees (not just trees of a fixed finite degree), and the transition function is graded, i.e., instead of a Boolean combination of direction-state pairs, it specifies a Boolean combination of distribution operations. There are two distribution operations: $\diamond(q_1, \dots, q_k)$ and its dual $\square(q_1, \dots, q_k)$. Intuitively, $\diamond(q_1, \dots, q_k)$ specifies that the automaton picks k different sons s_1, \dots, s_k of the current node and, for each $i \leq k$, sends a copy in state q_i to son s_i . Note that the states q_1, \dots, q_k are not necessarily all different.

A GHTA A is a tuple $\langle \Sigma, Q, q_0, \delta, \langle G, B \rangle, \langle \text{part}, \text{type}, \preceq \rangle \rangle$ where all elements but δ are defined as for AHTA, and $\delta : Q \times \Sigma \rightarrow \mathbb{B}^+(\diamond_Q \cup \square_Q)$ is a transition function that maps a state and an input letter to a positive Boolean combination of elements in $\diamond_Q = \{\diamond(q_1, \dots, q_k) \mid (q_1, \dots, q_k) \in Q^k, k \in \mathbb{N}\}$ and $\square_Q = \{\square(q_1, \dots, q_k) \mid (q_1, \dots, q_k) \in Q^k, k \in \mathbb{N}\}$.

We show how to define the run of a GHTA A on a Σ -labeled finitely-branching tree $T = \langle T, V \rangle$ by (locally) unfolding every \diamond_Q and \square_Q in $\delta(q, V(t))$ into a formula in $\mathbb{B}^+([d] \times Q)$ where d is the branching-degree of node t . For $k, d \in \mathbb{N}$, let $S(k, d)$ be the set of all ordered different k elements in $[d]$, i.e., $(s_1, \dots, s_k) \in S(k, d)$ iff for every $i \in [k]$ we have that $s_i \in [d]$, and that if $i \neq j$ then $s_i \neq s_j$. Observe that if $k > d$ then $S(k, d) = \emptyset$. For every $d \in \mathbb{N}$, define the function $\text{expand}_d : \mathbb{B}^+(\diamond_Q \cup \square_Q) \rightarrow \mathbb{B}^+([d] \times Q)$ that maps formula ϕ to the formula formed from ϕ by replacing every occurrence of a sub-formula of the form $\diamond(q_1, \dots, q_k)$ by the formula $\bigvee_{(s_1, \dots, s_k) \in S(k, d)} (\bigwedge(s_i, q_i))$, and every occurrence of a sub-formula of the form $\square(q_1, \dots, q_k)$ by the formula $\bigwedge_{(s_1, \dots, s_k) \in S(k, d)} (\bigvee(s_i, q_i))$. Observe that if $k > d$ then $\diamond(q_1, \dots, q_k)$ becomes the constant formula **false**, and $\square(q_1, \dots, q_k)$ becomes the constant formula **true**. The *run of a GHTA* A is defined as for an alternating tree automaton, except that one uses $\text{expand}_n(\delta(q, V(x)))$ instead of $\delta(q, V(x))$ for nodes x of T of degree n . Finally, the *hesitancy condition* defined above for AHTA is required to apply to the expanded transition function, i.e., insert the phrase “every $n \in \mathbb{N}$,” before the phrase “and every $\sigma \in \Sigma$ ”, and in items (i)-(iii) replace $\delta(q, \sigma)$ by $\text{expand}_n(\delta(q, \sigma))$. Acceptance is as for AHTA.

Lemma 2. *The emptiness problem for GHTA A over trees of branching degree at most d is decidable in time $2^{O(d \cdot |Q|^3)}$, where Q is the state set of A .*

Proof. Given a GHTA A with state set Q , convert it into an AHTA A' with the same state space by using the function expand_d defined above to transform its transition relation into a non-graded one. This is possible since we assumed a bound d on the branching degree of the input trees, and thus the transformation expand_d can be used in advance. This construction takes time that is $2^{O(|Q| \log d)}$. Recall that AHTA are a special case of alternating parity tree automata (APTA) with 3 priorities. Now apply the fact that the emptiness problem for APTA with p priorities over d -ary trees can be solved in time $2^{O(d \cdot |Q|^p)}$ [12].

4 From GCTL* to Graded Hesitant Automata

Elegant and optimal algorithms for solving the satisfiability and model-checking problems of CTL* were given using the automata-theoretic approach for branching-

time temporal logics [20]. Using this approach, one reduces satisfiability to the non-emptiness problem of a suitable tree automaton accepting all tree-models of a given temporal logic formula. We follow the same approach here, by reducing the satisfiability problem of GCTL* to the non-emptiness problem of GHTA. By Theorem 1, a GCTL* formula is satisfiable (in some, possibly infinite, labeled transition system) iff it has a finitely branching (though possibly unboundedly branching) tree model, which exactly falls within the abilities of GHTA. Our main technical result states that every GCTL* formula can be compiled into an exponentially larger GHTA (the rest of this section provides the proof):

Theorem 3. *Given a GCTL* formula ϑ , one can build a GHTA A_ϑ that accepts all the finitely-branching tree-models of ϑ . Moreover, A_ϑ has $2^{O(|\vartheta| \cdot \deg(\vartheta))}$ states, depth $O(|\vartheta|)$, and transition function of size $2^{O(|\vartheta| \cdot \deg(\vartheta))}$.*

An important observation that allows us to achieve an optimal construction is the following. Suppose that the formula $E^{\geq g}\psi$ holds at some node w of a tree. Then, by definition, there are at least g different paths $\rho'^1, \dots, \rho'^g \in \min(Con(S, w, \psi))$. Look at any g infinite extensions ρ^1, \dots, ρ^g of these paths in the tree, and note that by the definition of ψ -conservativeness all these extensions must satisfy ψ . Also observe that for every $i \neq j$, the fact that ρ'^i, ρ'^j are different and minimal implies that the longest common prefix ρ'^{ij} of ρ^i and ρ^j is not ψ -conservative. As it turns out, the other direction is also true, i.e., if there are g infinite paths ρ^1, \dots, ρ^g satisfying ψ , such that for every $i \neq j$ the common prefix ρ'^{ij} is not ψ -conservative, then there are g prefixes ρ'^1, \dots, ρ'^g of ρ^1, \dots, ρ^g respectively, such that $\rho'^1, \dots, \rho'^g \in \min(Con(S, w, \psi))$. Note that this allows us to reason about the cardinality of the set $\min(Con(S, w, \psi))$, by considering only the infinite paths ρ^1, \dots, ρ^g and their common prefixes, without actually looking at the minimal ψ -conservative paths ρ'^1, \dots, ρ'^g . In reality, we do not even have to directly consider the common prefixes ρ'^{ij} . Indeed, since the property of being ψ -conservative is upward closed (with respect to the prefix ordering \preceq of paths), showing that ρ'^{ij} is not ψ -conservative can be done by finding any extension of ρ'^{ij} that is not ψ -conservative. The following proposition formally captures this.

Proposition 1 *Given a GCTL* path formula ψ and a 2^{AP} -labeled tree $T = (T, V)$, then $T \models E^{\geq g}\psi$ iff there are g distinct nodes $y_1, \dots, y_g \in T$ (called breakpoints) such that for every $1 \leq i, j \leq g$ we have: (i) if $i \neq j$ then y_i is not a descendant of y_j ; (ii) the path from the root to the father x_i of y_i is not ψ -conservative; (iii) there is an infinite path ρ^i in T , starting at the root and going through y_i , such that $\rho^i \models \psi$.*

We are in a position to describe our construction of a GHTA accepting all finitely-branching tree-models of a given GCTL* formula. Naturally, the main difficulty lies in handling the graded modalities. The basic intuition behind the way our construction handles formulas of the form $\varphi = E^{\geq g}\psi$ is the following. Given an input tree, the automaton A_φ for this formula has to find at least g minimal ψ -conservative paths. At its core, A_φ runs g pairs of copies of itself in parallel. The reason these copies are not run independently is to ensure that

the two members of each pair are kept coordinated, and that different pairs do not end up making the same guesses (and thus overcounting the number of minimal ψ -conservative paths). The task of each of the g pairs is to detect some minimal ψ -conservative path that contributes 1 to the count towards g . This is done indirectly by using the characterization given by Proposition 1. Since this proposition requires checking if certain paths satisfy ψ , the automaton A_φ will access certain classic NBWs. We begin by establishing the existence of these:

Theorem 4. *Given an LTL formula ζ , there is an NBW A_ζ (resp. NFW B_ζ), both of size $2^{O(\zeta)}$, accepting exactly all infinite (resp. finite) words that satisfy ζ .*

Lemma 3. *Given an LTL formula ζ , there is an NBW A^ζ (of size $2^{O(\zeta)}$) such that A^ζ accepts a word w iff $w \models \zeta$, or $u \models \zeta$ for a prefix u of w . Moreover, A^ζ has an accepting sink \top , such that if r_0, r_1, \dots is an accepting run of A^ζ on w , and $i \geq 0$ satisfies $r_i \neq \top$, then a (finite or infinite) prefix u of w , of length $|u| > i$, satisfies ζ , and vice-versa (i.e., if a prefix u of w satisfies ζ , then there is an accepting run on w with $r_i \neq \top$ for all $i < |u|$).*

We can now finish the intuitive description of the construction of the automaton A_φ associated with a formula $\varphi = E^{\geq g}\psi$. Let Ψ be the LTL formula resulting from applying Lemma 1 to ψ . In essence, A_φ guesses the g descendants y_1, \dots, y_g of the root of the input tree as given in Proposition 1. For every $1 \leq i \leq g$, the automaton uses one copy of $A^{-\Psi}$ to verify that the path π , from the root to the father of y_i , is not ψ -conservative (by guessing some finite or infinite extension $\pi \preccurlyeq \pi'$ of it such that $\pi' \models \neg\Psi$), and one copy of A_Ψ to guess an infinite path π'' from the root through y_i such that $\pi'' \models \Psi$ (and is thus ψ -conservative).

4.1 The construction of GHTA A_ϑ for a GCTL* formula ϑ .

We induct on the structure of ϑ . Given a state sub-formula ϕ of ϑ (possibly including ϑ), for every formula $\theta \in \overline{\max(\phi)}$, let $A_\theta = \langle \Sigma, Q^\theta, q_0^\theta, \delta^\theta, \langle G^\theta, B^\theta \rangle, \langle \text{part}^\theta, \text{type}^\theta, \preceq^\theta \rangle \rangle$ be a GHTA accepting the finitely-branching tree-models of θ . The proof of correctness plus the definition of the hesitancy structure, i.e., of $\langle \text{part}^\theta, \text{type}^\theta, \preceq^\theta \rangle$, is in the full version (recall that the hesitancy structure is only used to decide in a space-efficient way membership, which is needed for our result that model-checking of GCTL* is in PSPACE). We build the GHTA A_ϕ accepting all finitely-branching tree-models of ϕ by suitably composing the automata of its maximal sub-formulas and their negations. Note that when composing these automata, we assume w.l.o.g. that the states of any occurrence of a constituent automaton of a sub-formula are disjoint from the states of any other occurrence of a constituent automaton (of the same or of a different sub-formula), as well as from any newly introduced states.⁵ Formally:

⁵ For example, when building an automaton for $\phi = \varphi_0 \vee \varphi_1$, in the degenerate case that $\varphi_0 = \varphi_1$ then A_{φ_1} is taken to be a copy of A_{φ_0} with its states renamed to be disjoint from those of A_{φ_0} . Also, the new state q_0 may be renamed to avoid a collision with any of the other states.

1. If $\phi = p \in AP$, then $A_\phi = \langle \Sigma, \{q\}, q, \delta, \langle \emptyset, \emptyset \rangle, \langle \text{part, type, } \preceq \rangle \rangle$ where $\delta(q, \sigma) = \text{true}$ if $p \in \sigma$ and false otherwise.
2. If $\phi = \varphi_0 \vee \varphi_1$ then A_ϕ is obtained by nondeterministically invoking either A_{φ_0} or A_{φ_1} . Thus, $A_\phi = \langle \Sigma, \bigcup_{i=0,1} Q^{\varphi_i} \cup \{q_0\}, q_0, \delta, \langle \bigcup_{i=0,1} G^{\varphi_i}, \bigcup_{i=0,1} B^{\varphi_i} \rangle, \beta \rangle$, where $\beta = \langle \text{part, type, } \preceq \rangle$, and for every $i \in \{0, 1\}$, every $\sigma \in \Sigma$, and every $q \in Q^{\varphi_i}$ we have that: $\delta(q, \sigma) = \delta^{\varphi_i}(q, \sigma)$, and $\delta(q_0, \sigma) = \delta^{\varphi_0}(q_0^{\varphi_0}, \sigma) \vee \delta^{\varphi_1}(q_0^{\varphi_1}, \sigma)$.
3. If $\phi = \neg\varphi$, then A_ϕ is obtained by dualizing the automaton A_φ . Formally, the *dual of a GHTA A* is the GHTA obtained by dualizing the transition function of A (i.e., switch \vee and \wedge , switch \top and \perp , and switch \square and \diamond), replacing the acceptance condition $\langle G, B \rangle$ with $\langle B, G \rangle$ (and toggling types).

Finally we deal with the case that $\phi = E^{\geq g}\psi$. Observe that ψ is a path formula and, by Lemma 1, reasoning about ψ can be reduced to reasoning about the LTL formula Ψ whose atoms are elements of $\max(\psi)$. Let $\Sigma' = 2^{\max(\psi)}$. By Theorem 4, there is an NBW $A_\Psi = \langle \Sigma', Q^+, q_0^+, \delta^+, G^+ \rangle$ accepting all infinite words in Σ'^ω satisfying Ψ . By Lemma 3, there is an NBW $A^{-\Psi} = \langle \Sigma', Q^-, q_0^-, \delta^-, G^- \rangle$ accepting all infinite words in Σ'^ω that either satisfy $\neg\Psi$ or have a prefix that does. Note that the states of these automata are denoted Q^+ and Q^- . We let A_ϕ be $\langle \Sigma, Q, q_0, \delta, \langle G, B \rangle, \langle \text{part, type, } \preceq \rangle \rangle$, whose structure we now define.

The set of states. $Q = Q_1 \cup Q_2$, where $Q_1 = (Q^+ \cup \{\perp\})^g \times (Q^- \cup \{\perp\})^g \setminus \{\perp\}^{2g}$, and $Q_2 = \bigcup_{\theta \in \overline{\max(\psi)}} Q^\theta$. The Q_1 states are used to run g copies of $A^{-\Psi}$ and g copies of A_Ψ in parallel. Every state in Q_1 is a vector of $2g$ coordinates where coordinates $1, \dots, g$ (called Ψ coordinates) contain states of A_Ψ , and coordinates $g+1, \dots, 2g$ (called $\neg\Psi$ coordinates) contain states of $A^{-\Psi}$. In addition, each coordinate may contain the special symbol \perp indicating that it is *disabled*, as opposed to *active*. We disallow the vector $\{\perp\}^{2g}$ with all coordinates disabled. States in Q_2 are all those from the automata A_θ for every maximal state subformula of ψ , or its negation. These are used to run A_θ whenever A_ϕ guesses that θ holds at a node. Also, for every $1 \leq i \leq g$, we denote by $Q_{\text{single}}^i = \{(q_1, \dots, q_{2g}) \in Q_1 \mid q_i \neq \perp, \text{ and for all } j \leq g, \text{ if } j \neq i \text{ then } q_j = \perp\}$ the set of all states in Q_1 in which the only active Ψ coordinate is i .

The initial state. $q_0 = (q_1, \dots, q_{2g})$ where for every $1 \leq i \leq g$ we have that $q_i = q_0^+$ and for every $g+1 \leq i \leq 2g$ we have that $q_i = q_0^-$.

The acceptance condition. $B = \bigcup_{\theta \in \overline{\max(\psi)}} B^\theta$ and $G = G' \cup G'' \cup (\bigcup_{\theta \in \overline{\max(\psi)}} G^\theta)$, where $G' = \{(q_1, \dots, q_{2g}) \in Q_{\text{single}}^i \mid q_i \in G^+\}$ is the set of all states in Q_1 in which the only active Ψ coordinate contains a good state, and $G'' = \{(q_1, \dots, q_{2g}) \in Q_1 \mid \forall i. 1 \leq i \leq g \rightarrow q_i = \perp, \text{ and } \exists j. g+1 \leq j \leq 2g \wedge q_j \in G^-\}$ is the set of all states in Q_1 in which all the Ψ coordinates are inactive, and some $\neg\Psi$ coordinate contains a good state.

The transition function. δ is defined, for every $\sigma \in \Sigma$, as follows:

- For every $q \in Q_2$, let $\theta \in \overline{\max(\psi)}$ be such that $q \in Q^\theta$, and define $\delta(q, \sigma) = \delta^\theta(q, \sigma)$. I.e., for states in Q_2 , follow the rules of their respective automata.
- For every $q \in Q_1$, we define $\delta(q, \sigma) := \bigvee_{\sigma' \in \Sigma'} (J \wedge K \wedge L)$ where $J = \bigvee_{X \in \text{Legal}(q, \sigma')} \diamond(X)$, $K = \bigwedge_{\theta \in \sigma'} \delta^\theta(q_0^\theta, \sigma)$, $L = \bigwedge_{\theta \notin \sigma'} \delta^{-\theta}(q_0^{-\theta}, \sigma)$, where $\text{Legal}(q, \sigma')$ is the set of all *legal distributions* of (q, σ') , and is defined later.

Informally, the disjunction $\bigvee_{\sigma' \in \Sigma'}$ corresponds to all possible guesses of the set of maximal subformulas of ψ that currently hold. Once a guess σ' is made, the copies of $\mathbb{A}^{-\Psi}$ and \mathbb{A}_{Ψ} simulated by the states appearing in $\text{Legal}(q, \sigma')$ proceed as if the input node was labeled by the letter σ' . The conjunction $(\wedge_{\theta \in \sigma'} \delta^\theta(q_0^\theta, \sigma)) \wedge (\wedge_{\theta \notin \sigma'} \delta^{-\theta}(q_0^{-\theta}, \sigma))$ ensures that a guess is correct by launching a copy of \mathbb{A}_θ for every subformula $\theta \in \sigma'$ that was guessed to hold, and a copy of $\mathbb{A}_{-\theta}$ for every subformula θ guessed not to hold.

We define *legal distribution*. Intuitively, a legal distribution of (q, σ') is a sequence q^1, \dots, q^m of different states from Q_1 that “distribute” among them, without duplication, the coordinates active in q , while making sure that for every $1 \leq i \leq g$ coordinate i (which simulates a copy of \mathbb{A}_{Ψ}) does not get separated from the coordinate $i + g$ (which simulates its partner copy of $\mathbb{A}^{-\Psi}$) for as long as i is not the only active Ψ coordinate. As expected, every active coordinate j , in any of the states q^1, \dots, q^m , follows from q_j by using the transitions available in the automaton it simulates: \mathbb{A}_{Ψ} if $j \leq g$, or $\mathbb{A}^{-\Psi}$ if $j > g$.

More formally, given a letter $\sigma' \in \Sigma'$, and a state $q = (q_1, \dots, q_{2g}) \in Q_1$ in which the active coordinates are $\{i_1, \dots, i_k\}$, we say that a sequence $X = q^1, \dots, q^m$ (for some $m \geq 1$) of distinct states in Q_1 is a *legal distribution* of (q, σ') if the following conditions hold: (i) the coordinates active in the states q^1, \dots, q^m are exactly i_1, \dots, i_k , i.e., $\{i_1, \dots, i_k\} = \cup\{i \in \{1, \dots, 2g\} \mid \exists 1 \leq l \leq m \text{ s.t. } q_i^l \neq \perp\}$. (ii) if a coordinate i_j is active in some $q' \in X$ then it is not active in any other $q'' \in X$; (iii) if $1 \leq i_j < i_l \leq g$ are two active Ψ coordinates in some $q' \in X$, then $q'_{i_j+g}, q'_{i_l+g} \in Q^- \setminus \{\top\}$, i.e., the coordinates $i_j + g, i_l + g$ are also active in q' and do not contain the accepting sink of $\mathbb{A}^{-\Psi}$; (iv) if i_j is active in some $q' \in X$ then $(q_{i_j}, \sigma', q'_{i_j}) \in \delta^+$ if $i_j \leq g$, and $(q_{i_j}, \sigma', q'_{i_j}) \in \delta^-$ if $i_j > g$. I.e., active Ψ coordinates evolve according to the transitions of \mathbb{A}_{Ψ} , and active $-\Psi$ coordinates according to the those of $\mathbb{A}^{-\Psi}$.

Remark 1. We make two observations. First, the $2g$ copies of $\mathbb{A}^{-\Psi}$ and \mathbb{A}_{Ψ} can not simply be launched from the root of the tree using a conjunction in the transition relation. The reason is that if this is done then there is no way to enforce property (i) of Proposition 1. Second, a cursory look may suggest that different copies of $\mathbb{A}^{-\Psi}$ and \mathbb{A}_{Ψ} that are active in the current vector may be merged. Unfortunately, this cannot be done since $\mathbb{A}^{-\Psi}$ and \mathbb{A}_{Ψ} are nondeterministic, and thus, different copies of these automata must be able to make independent guesses in the present in order to accept different paths in the future.

Proposition 2 *The automaton \mathbb{A}_ϑ is a GHTA with depth $O(|\vartheta|)$ and $2^{O(|\vartheta| \cdot \deg(\vartheta))}$ many states, and the size of its transition function is $2^{O(|\vartheta| \cdot \deg(\vartheta))}$.*

5 Complexity of Satisfiability and MC of GCTL*

Theorem 5. *A satisfiable GCTL* formula ϑ has a tree model of branching degree at most $2^{O(|\vartheta| \cdot \deg(\vartheta))}$.*

Proof. Suppose ϑ is satisfiable. By Theorem 1, ϑ has a finitely-branching tree model. Observe, by Theorem 3, that $|Q| = 2^{O(|\vartheta|\cdot\deg(\vartheta))}$, where Q is the state set of the automaton A_ϑ defined in that proof. Hence, it is enough to prove that every tree model of ϑ has a subtree of branching degree $|Q|^2$ that also models ϑ .

To prove this claim, we use the membership game G_{T,A_ϑ} of the input tree T and the automaton A_ϑ . There are two players, *automaton* and *pathfinder*. Player automaton moves by resolving disjunctions in the transition relation of A_ϑ , and is trying to show that T is accepted by A_ϑ . Player pathfinder moves by resolving conjunctions, and is trying to show that T is not accepted by A_ϑ . The game uses auxiliary tree structured arenas to resolve each transition of the automaton. This is a simple case of a *hierarchical parity game* [3]. As usual, player automaton has a winning strategy if and only if $T \models A_\vartheta$. By memoryless determinacy of parity games on infinite arenas, player automaton has a winning strategy if and only if he has a memoryless winning strategy. For a fixed memoryless strategy str , one can prove, by looking at the transition function of A_ϑ , that every play consistent with str , and every node t of the input tree T , only visits at most $|Q|^2$ sons of t , thus inducing a subtree which is the required boundedly-branching tree model.

Theorem 6. *The satisfiability problem for GCTL* over LTSs is 2EXPTIME-COMPLETE, and model checking GCTL* for finite LTSs is PSPACE-COMPLETE.*

Proof. The lower-bounds already hold for CTL*. Theorems 3, 5 and Lemma 2 give the upper-bound for satisfiability. For the upper-bound for model checking, given an LTS S (with largest degree d), and a GCTL* formula ϑ , using Theorem 3 construct the GHTA $A_{\neg\vartheta}$, which has $2^{O(|\vartheta|\cdot\deg(\vartheta))}$ states, transition function of size $2^{O(|\vartheta|\cdot\deg(\vartheta))}$, and depth $O(|\vartheta|)$. As in the proof of Lemma 2, build an equivalent AHTA A' of size $d + (2^{O(|\vartheta|\cdot\deg(\vartheta))} \cdot |Q|^d) + 2^{O(|\vartheta|\cdot\deg(\vartheta))} = 2^{O(|\vartheta|\cdot\deg(\vartheta)+d\cdot|\vartheta|\cdot\deg(\vartheta))}$, and of depth $\partial = O(|\vartheta|)$. By Theorem 2, the membership problem of the AHTA A' on S can be solved in space $O(\partial \log^2(|S| \cdot ||A'||))$ which is polynomial in $|\vartheta|$ and $|S|$ (using $\deg(\vartheta) \leq |\vartheta|$ and $d \leq |S|$).

6 Discussion

This work shows that GCTL* is an expressive logic (it is equivalent, over trees, to MPL and can express fairness and counting over paths) whose satisfiability and model-checking problems have the same complexity as that of CTL*.

GCTL* was defined in [5]. However, only the fragment GCTL was studied. As the authors note in the conference version of that paper, their techniques, that worked for GCTL, do not work for GCTL*. Moreover, they also suggested a line of attack that does not seem to work; indeed, it was left out of the journal version of their paper [6]. Instead, our method is a careful combination of the automata-theoretic approach to branching-time logics [20], a characterization of the graded path modality (Proposition 1), and a boundedly-branching tree model property whose proof uses game-theoretic arguments (Theorem 5). Moreover, our technique immediately recovers the main results about GCTL from [5], i.e., satisfiability for GCTL is ExpTIME-COMPLETE and the model checking

problem for GCTL is in PTIME (Indeed, consider the construction in Theorem 3 of A_ϑ when ϑ it taken from the fragment GCTL of GCTL*, and in particular where it comes to a subformula ϕ of the form $\phi = E^{\geq g}\psi$. Since ψ is either of the form pUq or Xp , the number of new states added at this stage is a constant. Thus, the number of states of A_ϑ is linear in the size of ϑ). In other words, our technique suggests a powerful new way to deal with graded path modalities.

When investigating the complexity of a logic with a form of counting quantifiers, one must decide how the numbers in these quantifiers contribute to the length of a formula, i.e., to the input of a decision procedure. In this paper we assume that these numbers are coded in unary, rather than binary. There are a few reasons for this. First, the unary coding naturally appears in description and predicate logics [8]. As pointed out in [19], this reflects the way in which many decision procedures for these logics work: they explicitly generate n individuals for $\exists^{\geq n}$. Second, although the complexity of the binary case is sometimes the same as that of the unary case, the constructions are significantly more complicated, and are thus much harder to implement [6, 7]. At any rate, as the binary case is useful in some circumstances we plan to investigate this in the future.

Comparison with (some) other approaches. Although showing that satisfiability of GCTL* is decidable is not hard (for example, by reducing to MSOL), identifying the exact complexity is much harder. Indeed, there is no known satisfiability-preserving translation of GCTL* to another logic that would yield the optimal 2EXPTIME upper bound. We discuss two such candidate translations. First, in this article we show a translation from GCTL* to MPL. Unfortunately, the complexity of satisfiability of MPL is non-elementary. Second, there is no reason to be optimistic that a translation from GCTL* to $G\mu$ -calculus (whose satisfiability is EXPTIME-COMPLETE) would yield the optimal complexity since a) already the usual translation from CTL* to μ -calculus does not yield optimal complexity [13], and b) the translation given in [6] from GCTL to $G\mu$ -calculus does not yield optimal complexity. Moreover, the usual translation from CTL* to μ -calculus uses automata, and thus automata for GCTL* (from which we get our results directly) have to be developed anyway.

Future work. Recall that the graded μ -calculus was used to solve questions (such as satisfiability) for the description logic $\mu\mathcal{ALCQ}$ [7]. Similarly, our techniques for GCTL* might be useful for solving questions in \mathcal{ALCQ} combined with temporal logic, such as for the graded extension of CTL* $_{\mathcal{ALC}}$ [17]. Second, the GCTL model checking algorithm from [6] has been implemented in the NuSMV model-checker to provide more than one counter-example when a GCTL formula is not satisfied. We are thus optimistic that existing CTL* model-checkers can be fruitfully extended to handle GCTL*.

References

1. S. Almagor, U. Boker, and O. Kupferman. What's decidable about weighted automata? In *Intl. Symp. on Automated Technology for Verification and Analysis*, volume 6996 of *LNCS*, pages 482–491, 2011.

2. B. Aminof, O. Kupferman, and R. Lampert. Rigorous approximated determinization of weighted automata. In *Symp. on Logic in Computer Science*, pages 345–354, 2011.
3. B. Aminof, O. Kupferman, and A. Murano. Improved model checking of hierarchical systems. *Information and Computation*, 210:68–86, 2012.
4. M. Arenas, P. Barceló, and L. Libkin. Combining Temporal Logics for Querying XML Documents. In *ICDT*, LNCS 4353, pages 359–373, 2007.
5. A. Bianco, F. Mogavero, and A. Murano. Graded computation tree logic. In *Symp. on Logic in Computer Science*, pages 342–351. IEEE, 2009.
6. A. Bianco, F. Mogavero, and A. Murano. Graded computation tree logic. *ACM Transactions on Computational Logic*, 13(3):25, 2012.
7. P.A. Bonatti, C. Lutz, A. Murano, and M.Y. Vardi. The Complexity of Enriched Mu-Calculi. *Logical Methods in Computer Science*, 4(3):1–27, 2008.
8. D. Calvanese, G. De Giacomo, and M. Lenzerini. Reasoning in expressive description logics with fixpoints based on automata on infinite trees. In *International Joint Conference on Artificial Intelligence*, pages 84–89, 1999.
9. M. de Rijke. A note on graded modal logic. *Studia Logica*, 64(2):271–283, 2000.
10. M. Droste, W. Kuich, and H. Vogler. *Handbook of Weighted Automata*. Springer, 2009.
11. C. Eisner, D. Fisman, J. Havlicek, Y. Lustig, A. McIsaac, and D. V. Campenhoult. Reasoning with temporal logic on truncated paths. In *Computer Aided Verification*, LNCS 2725, pages 27–39, 2003.
12. E.A. Emerson and C.S. Jutla. The complexity of tree automata and logics of programs. *SIAM Journal of Computing*, 29(1):132–158, 1999.
13. E.A. Emerson and A.P. Sistla. Deciding branching time logic. In *Symposium on Theory of Computing*, pages 14–24, 1984.
14. A. Ferrante, A. Murano, and M. Parente. Enriched μ -calculi module checking. *Logical Methods in Computer Science*, 4(3), 2008.
15. A. Ferrante, M. Napoli, and M. Parente. Model Checking for Graded CTL. *Fundamenta Informaticae*, 96(3):323–339, 2009.
16. K. Fine. In So Many Possible Worlds. *Notre Dame Journal of Formal Logic*, 13:516–520, 1972.
17. V. Gutiérrez-Basulto, J.C. Jung, and C. Lutz. Complexity of branching temporal description logics. In *Euro. Conf. on Artificial Intelligence*, pages 390–395, 2012.
18. T.A. Henzinger. Quantitative reactive modeling and verification. *Comput. Sci.*, 28(4):331–344, November 2013.
19. O. Kupferman, U. Sattler, and M.Y. Vardi. The Complexity of the Graded μ -Calculus. In *Conference on Automated Deduction*, LNCS 2392, pages 423–437. Springer, 2002.
20. O. Kupferman, M.Y. Vardi, and P. Wolper. An Automata Theoretic Approach to Branching-Time Model Checking. *Journal of the ACM*, 47(2):312–360, 2000.
21. V. Malvone, F. Mogavero, A. Murano, and L. Sorrentino. On the counting of strategies. In *Int. Symp. on Temporal Representation and Reasoning*, 2015.
22. F. Moller and A. Rabinovich. Counting on CTL*: On the expressive power of monadic path logic. *Information and Computation*, 184(1):147–159, 2003.
23. S. Tobies. PSPACE Reasoning for Graded Modal Logics. *Journal of Logic and Computation*, 11(1):85–106, 2001.
24. W. van der Hoek and J.J.Ch. Meyer. Graded modalities in epistemic logic. In *Symp. on Logical Foundations of Computer Science*, pages 503–514, 1992.
25. M. Y. Vardi and P. Wolper. Reasoning about infinite computations. *Information and Computation*, 115(1):1–37, 1994.

Verification of Asynchronous Mobile-Robots in Partially-Known Environments^{*}

Benjamin Aminof¹ and Aniello Murano² and Sasha Rubin² and Florian Zuleger¹

¹ Technische Universität Wien

² Università degli Studi di Napoli "Federico II"

Abstract. This paper establishes a framework based on logic and automata theory in which to model and automatically verify that multiple mobile robots, with sensing abilities, moving asynchronously, correctly perform their tasks. The motivation is from practical scenarios in which the environment is not completely known to the robots, e.g., physical robots exploring a maze, or software agents exploring a hostile network. The framework shows how to express tasks in a logical language, and exhibits an algorithm solving the *parameterised* verification problem, where the graphs are treated as the parameter. The main assumption that yields decidability is that the robots take a bounded number of turns. We prove that dropping this assumption results in undecidability, even for robots with very limited ("local") sensing abilities.

1 Introduction

Autonomous mobile robots are designed to achieve some task in an environment without a central control. Foundational tasks include, for example, rendezvous (gather all robots in a single position) and reconfiguration (move to a new configuration in a collision-free way) [25, 14, 15]. This paper studies robots in *partially known* environments, i.e., robots do not have global information about the environment, but may know some (often topological) information (e.g., whether the environment is connected, or that it is a ring of some unknown size) [14]. The motivation for studying partially known environments is that in many practical scenarios the robots are unaware of the exact environment in which they are operating, e.g., mobile software exploring a hostile computer network, or physical robots that rendezvous in an environment not reachable by humans.

To illustrate, here is an *example reconfiguration problem*. Suppose that k robots find themselves on different internal nodes of a binary tree, and each robot has to reach a different leaf in a collision free way. Each robot can sense

* Benjamin Aminof and Florian Zuleger were supported by the Austrian National Research Network S11403-N23 (RiSE) of the Austrian Science Fund (FWF) and by the Vienna Science and Technology Fund (WWTF) through grant ICT12-059. Aniello Murano was supported by FP7 EU project 600958-SHERPA. Sasha Rubin is a Marie Curie fellow of the Istituto Nazionale di Alta Matematica.

if its left (or right) child is occupied by a robot. One protocol for solving this problem (assuming a large enough tree) is for each robot to execute ‘go to the left child, then repeatedly go the right child’ until a leaf is reached. Each move is guarded by a test that the child it wants to move to is not currently occupied.

As the example illustrates, we make the following modeling choices: environments are discrete (rather than continuous), robots have finite memory (rather than being oblivious, or being Turing powerful), robots are nondeterministic (rather than probabilistic), robots move asynchronously (rather than synchronously), and robots can sense the positions and the internal states of each robot no matter where they are, i.e., they can perform “remote” tests (but cannot leave information at visited positions).³ The assumption that robots move asynchronously is motivated as follows: processes in distributed systems have no common notion of time since they may be running with different internal clocks, and thus a standard abstraction is to assume that processes’ actions are interleaved, see [26]. There are two main ways to interleave the robots: an adversary tries to make sure that the robots do not succeed [16], and a co-operator tries to help the robots succeed (reminiscent of schedulers in strategic reasoning [7]).

In this paper we provide a framework for modeling and verifying that multiple mobile robots achieve a task (under adversarial and co-operative interleavings) in a partially-known environment. We now explain how we model the fact that environments are partially-known. Fix a class \mathcal{G} of environments (e.g., \mathcal{G} is the set of all lines, or all grids, or all rings). The *parameterised verification problem* states: Given robots \bar{R} , decide if they solve the task T on all graphs $G \in \mathcal{G}$. Requiring the robots to solve the task T on all graphs from a class \mathcal{G} is how we model that the robots operate in partially-known environments — the robots know they are in a graph from \mathcal{G} , but they do not know which one. In contrast, the classic (non-parameterised) verification problem states: Given robots \bar{R} and a graph $G \in \mathcal{G}$, decide if robots \bar{R} solve the task T on G . In that setting, the robots can be designed to exploit the exact structure (size, etc.) of the graph.

Aims and Contributions. The aim of this work is to provide a formal framework in which one can reason (both mathematically and computationally) about multiple mobile-robots, with sensing abilities, moving asynchronously in a discrete, static, partially-known environment. We prove that parameterised verification is undecidable already for line-environments, and even for robots with limited tasks (e.g., halting) which can only detect collisions. I.e., a robot can only sense which other robots share its current position. This undecidability result also holds for robots that move synchronously. On the other hand, we prove that parameterised verification is decidable for scenarios in which the number of times that the robots take turns is bounded.⁴ This decidability result is very robust:

³ The ability to sense positions is a form of vision, while the ability to sense internal states is a form of communication

⁴ In the example reconfiguration problem, there is some ordering of the robots that switches turns at most k times in which the stated robot-protocol succeeds. Also, for every ordering of the robots that switches turns a sufficiently large number of times, the protocol succeeds (“ordering” and “switching” are formalised in Section 3).

it holds on a very general class of graphs called *context-free sets of graphs* which include e.g., rings, trees, series-parallel graphs, but not grids; it also holds with very powerful abilities called *position-tests* which allow each robot to remotely test the positions of all the robots using formulas which include, e.g., the ability to test connectivity, as well as *state-tests* that allow each robot to remotely test the internal state of the other robots; it holds for tasks that are expressible using a new logic MRTL (Multiple-Robot Task Logic), which can express many natural tasks, e.g., reconfiguration, gathering, and “safe” variations (i.e., complete the task without touching dangerous positions).

Related Work. The work closest to ours is [30] which also considered the parameterised verification problem for multi-robot systems. However, in that paper, the decidability result (and the corresponding logic RTL) was only for one-robot systems (i.e., $k = 1$), and the undecidability result for $k = 2$ was for multiple robots that move *synchronously* and have *remote* tests.

The distributed-computing community has proposed and studied a number of models of robot systems, e.g., recently [24, 18, 10, 13, 8, 17]. This literature is mostly mathematical, and *theorems from this literature are parameterised*, i.e., they may involve graph-parameters (e.g., the maximum degree, the number of vertices, the diameter), memory parameters (e.g., the number of internal states of the robot protocol), and the number of robots may be a parameter. Only recently has there been emphasis on formal analysis of correctness of robots in a parameterised setting [22, 4, 27, 23, 30, 29]. In these formal analyses, typically it is the *number of agents* that is treated as the parameter [22, 4, 27, 23]. In contrast, in this paper (as in [30]) we apply formal methods to the parameterised verification problem in which it is the *environment that is parameterised*.

Also, the formal-verification community has placed emphasis on distributed models in which processes are stationary and interact by sending messages (e.g., in a broadcast or rendezvous manner) or by using guarded commands. The parameterised verification problem for such distributed processes is, in general, undecidable [31]. By simplifying the systems (e.g., restricting the mode of communication, the abilities of the processes, the specification languages, etc.) one can get decidable parameterised verification, recently e.g., [12, 1–3].

We refer the reader to [30, Section 7] for an up-to-date and detailed discussion of the connections between multi-robot systems, classic automata theory (i.e., graph-walking automata) and distributed systems (in particular, token-passing systems). Finally, we mention that there is a smattering of work on parameterised *synthesis* (called *generalised planning* in the AI literature) [11, 19–21].

2 Background: Automata Theory

Write B^* and B^ω for the sets of finite and infinite sequences over alphabet B , respectively. The empty sequence is denoted ϵ . Write $[n]$ for the set $\{1, 2, \dots, n\}$.

Graphs and Trees. A Σ -*graph*, or *graph*, G , is a tuple (V, E, Σ, λ) where V is a finite set of *vertices*, $E \subseteq V \times V$ is the *edge relation*, Σ is a finite set of *edge*

labels, and $\lambda : E \rightarrow \Sigma$ is the *edge labeling function*. A Δ -ary tree (for $\Delta \in \mathbb{N}$) is a Σ -graph (V, E, Σ, λ) where (V, E) is a tree, $\Sigma = [\Delta] \cup \{\text{up}\}$, and λ labels the edge leading to the node in direction i (if it exists) by i , and the edge leading to the parent of a node (other than the root) is labelled by *up*. We may rename the labels for convenience, e.g., for binary trees ($\Delta = 2$) we let $\Sigma = \{\text{lc}, \text{rc}, \text{up}\}$ where *lc* replaces 1 and *rc* replaces 2.

Monadic Second-order Logic. Formulas are interpreted in Σ -graphs G . Define the set of monadic second-order formulas $\text{MSOL}(\Sigma)$ as follows. Formulas of $\text{MSOL}(\Sigma)$ are built using *first-order variables* x, y, \dots that vary over vertices, and *set variables* X, Y, \dots that vary over sets of vertices. The *atomic formulas* (when interpreted over Σ -graphs) are: $x = y$ (denoting that vertex x is the same as vertex y), $x \in X$ (denoting that vertex x is in the set of vertices X), $\text{edg}_\sigma(x, y)$ (denoting that there is an edge from x to y labeled $\sigma \in \Sigma$) and *true* (the formula that is always true). The formulas of $\text{MSOL}(\Sigma)$ are built from the atomic formulas using the Boolean connectives (i.e., $\neg, \vee, \wedge, \rightarrow$) and variable quantification (i.e., \forall, \exists over both types of variables). A variable that is not quantified is called *free*. The fragment of $\text{MSOL}(\Sigma)$ which does not mention set variables is called *first-order logic*, denoted $\text{FOL}(\Sigma)$. Write $\text{MSOL}_k(\Sigma)$ for formulas with at most k many free first-order variables and no free set-variables. We abbreviate z_1, \dots, z_k by \bar{z} . We write $\phi(x_1, \dots, x_k)$ to mean that the free variables from the formula ϕ are amongst the set $\{x_1, \dots, x_k\}$ – note that the formula $\phi(x_1, \dots, x_k)$ does not need to use all of the variables x_1, \dots, x_k . For a graph G , and $v_1, \dots, v_k \in V$, we write $G \models \phi(v_1, \dots, v_k)$ to mean that ϕ holds in G with variable x_i simultaneously substituted by vertex v_i (for all $i \in [k]$ for which x_i occurs free in ϕ). Here are some examples of formulas and their meanings: The formula $\forall x(x \in X \rightarrow x \in Y)$ means that $X \subseteq Y$. Similarly, there are formulas for the set operations $\cup, \cap, =$, and relative complement $X \setminus Y$. The formula $\text{edg}(x, y) := \bigvee_{\sigma \in \Sigma} \text{edg}_\sigma(x, y)$ means that there is an edge from x to y (here Σ is assumed to be finite). The formula $E^*(x, y) := \forall Z[(\text{closed}_E(Z) \wedge x \in Z) \rightarrow y \in Z]$ where $\text{closed}_E(Z)$ is $\forall a \forall b[(a \in Z \wedge E(a, b)) \rightarrow b \in Z]$ defines the transitive closure of E . Generally, MSOL can express the 1-ary transitive closure operator (e.g., [30]).

The Validity Problem and Courcelle's Theorem. A *sentence* is a formula with no free variables. Let Φ be a set of sentences, and let \mathcal{G} be a set of graphs. The Φ -*validity problem* of \mathcal{G} is to decide, given $\phi \in \Phi$, whether for all graphs $G \in \mathcal{G}$, it holds that $G \models \phi$. Unfortunately, the $\text{MSOL}(\Sigma)$ -validity problem for the set \mathcal{G} of all Σ -graphs is undecidable. However, Courcelle's Theorem states that MSOL -validity of context-free sets of graphs is uniformly decidable, i.e., there is an algorithm that given a description of a context-free set of graphs \mathcal{G} and an MSOL -sentence ϕ decides if every graph in \mathcal{G} satisfies ϕ [9]. Context-free sets of graphs are the analogue of context-free sets of strings, and can be described by graph grammars, equations using certain graph operations, or MSOL -transductions of the set of trees. Formally, \mathcal{G} is *context-free* if it is MSOL -definable and of bounded clique-width [9]. Examples include, for a fixed alphabet, the set of labeled lines, rings, trees, series-parallel graphs, cliques, but not the set of grids.

Automata and Regular Expressions. Ordinary *regular-expressions* over a finite alphabet B are built from the sets \emptyset , $\{\epsilon\}$, and $\{b\}$ ($b \in B$), and the operations union $+$, concatenation \cdot , and Kleene-star $*$. Kleene’s Theorem states that the languages definable by regular expressions over alphabet B are exactly those recognised by finite automata over alphabet B . An ω -*regular expression* over alphabet B is inductively defined to be of the form: exp^ω , $exp \cdot r$, or $r + r'$, where exp is an ordinary regular-expression over B , and r, r' are ω -regular expressions over B . An ω -regular language is one defined by an ω -regular expression. A variation of Kleene’s Theorem says that the languages definable by ω -regular expressions over alphabet B are exactly the languages recognised by Büchi automata over alphabet B (which are like finite automata except they take infinite words as input, and accept if some accepting state occurs infinitely often).

3 The Model of Robot Systems

In this section we provide a framework for modeling multi-robot systems parameterised by their environment. Environments are modeled as Σ -graphs G and robots are modeled as regular languages of instructions. An instruction either tells the robot to move along an edge, or to test robot positions (e.g., a robot can learn which other robots are at the same vertex as it is, or if there is a robot north of it). Tests are formalised as logical formulas.

Instructions for Robots. Fix a number k of robots and a set of edge labels Σ . A *command* is a symbol from $\{\uparrow_\sigma: \sigma \in \Sigma\} \cup \{\circlearrowleft\}$. The command \uparrow_σ tells the robot to move from its current vertex along the edge labeled σ , and the command \circlearrowleft tells the robot to stay at its current vertex. A *position-test* is a formula from $\text{MSOL}_k(\Sigma)$. A *state-test* is an expression of the form “robot i is in state q ”, where $i \in [k]$ and q is a symbol denoting a state of a robot (formally we may assume that all robots have states from \mathbb{N} , and that $q \in \mathbb{N}$)⁵. A position test $\tau(x_1, \dots, x_k)$ allows the robot to test that $\tau(x_1, \dots, x_k)$ holds in G , where x_i is the current vertex of robot R_i in G . Simple tests include “ $x_i = x_j$ ” which tests if robots i and j are in the same vertex (i.e., collision detection), and “ $\text{edg}(x_i, x_j) \vee \text{edg}(x_j, x_i)$ ” which tests if robots i and j are adjacent. A *test* is a state-test or a position-test. The *instruction set* $\text{INS}_{\Sigma, k}$ consists of all expressions of the form $\tau \rightarrow \kappa$ where τ is a test and κ is a command.

Robots, Configurations, Runs. A k -*robot ensemble* is a vector of robots $\langle R_1, \dots, R_k \rangle$ where each *robot* $R_i = \langle Q_i, \delta_i \rangle$, each Q_i is a finite set of *states*, and each $\delta_i \subset Q_i \times \text{INS}_{\Sigma, k} \times Q_i$ is a finite *transition* relation. For technical convenience, we assume that robot i does not test its own state, i.e., no *ins* in a transition $(p, \text{ins}, q) \in \delta_i$ contains any occurrences of state-tests of the form “robot i is in state j ”. We designate a subset $I_i \subseteq Q_i$ of the states of robot i

⁵ Note that, for ease of exposition, we do not explicitly allow Boolean combinations of state-tests. However, these can be indirectly performed by chaining state-tests (remembering the previous test results in the local state) to perform conjunctions, and using nondeterminism for disjunctions.

as *initial* states, and a subset $A_i \subseteq Q_i$ of its states as *accepting* states. A state $p \in Q_i$ is called *halting* if the only transition the robot has from p is (p, true, p) . Thus we model a halting robot as one that forever stays in the same state and does not move. The halting states are denoted $H_i \subseteq Q_i$.

Fix a Σ -graph G . A *configuration* c of $\langle R_1, \dots, R_k \rangle$ on graph G is a pair $\langle \bar{v}, \bar{q} \rangle \in V^k \times \prod_{i \in [k]} Q_i$. A configuration is *initial* if $\bar{q} \in \prod I_i$. For a test τ and a configuration $c = \langle \bar{u}, \bar{p} \rangle$, define $c \Vdash \tau$ to mean that configuration c makes τ true in G . Formally, if τ is a position test then define $c \Vdash \tau$ iff $G \models \tau(\bar{u})$, and if τ is a state-test, say “robot i is in state j ”, then define $c \Vdash \tau$ iff $p_i = j$. The following definition of \vdash_i expresses that one configuration results from another after robot i successfully executes an instruction, while the rest are idle: for $i \in [k]$ and configurations $c = \langle \bar{w}, \bar{q} \rangle, d = \langle \bar{v}, \bar{p} \rangle$, write $c \vdash_i d$ if $p_j = q_j$ and $w_j = v_j$ for all $j \neq i$, and there exists a transition $(p_i, \tau \rightarrow \kappa, q_i) \in \delta_i$ (i.e., of robot R_i) such that $c \Vdash \tau$ (i.e., the current configuration satisfies the test τ) and, if $\kappa = \uparrow_\sigma$ then $\lambda(w_i, v_i) = \sigma$, and if $\kappa = \circlearrowleft$ then $w_i = v_i$.

Schedules and Runs. A *schedule* is a finite or infinite sequence $\mathcal{S} = s_1 s_2 s_3 \dots$ where $s_i \in [k]$. A *run* ρ of $\langle R_1, \dots, R_k \rangle$ on G starting with an initial configuration c according to schedule \mathcal{S} is a finite or infinite sequence $c_1 c_2 c_3 \dots$ of configurations such that $c_1 = c$ and for all i , $c_i \vdash_{s_i} c_{i+1}$. The *set (resp. sequence) of positions of a run* $\alpha = \langle \bar{v}_1, \bar{p}_1 \rangle \langle \bar{v}_2, \bar{p}_2 \rangle \dots$ is the set of positions $\{\bar{v}_1, \bar{v}_2, \dots\}$ (resp. sequence $\bar{v}_1 \bar{v}_2 \dots$ of positions) of its configurations. In a similar way define the *set (resp. sequence) of positions of robot i on a run*.

Orderings. A *(finite) k -ordering* is a string $\alpha \in [k]^+$, say of length $N + 1$, such that $\alpha_i \neq \alpha_{i+1}$ for $1 \leq i \leq N$. Write $\|\alpha\| = N$ to mean that $|\alpha| = N + 1$, and say that α is N -switching. E.g., 171 is 2-switching. Say that a schedule \mathcal{S} follows α if \mathcal{S} is in $\alpha_1^* \alpha_2^* \dots \alpha_N^* \alpha_{N+1}^*$ or $\alpha_1^* \alpha_2^* \dots \alpha_N^* \alpha_{N+1}^\omega$, i.e., robot α_1 is scheduled for some (possibly no) time, then robot α_2 is scheduled, and so on, until α_{N+1} which can be scheduled forever. Similarly, an *infinite k -ordering* of k robots is a string $\alpha \in [k]^\omega$ such that $\alpha_i \neq \alpha_{i+1}$ for all $i \in \mathbb{N}$. In this case write $\|\alpha\| = \infty$. A schedule follows α if the schedule is in the set $\alpha_1^* \alpha_2^* \dots$.

Robot Tasks. Robots should achieve some task in their environment. We give some examples of foundational robot tasks [25]: A robot ensemble *deploys* or *reconfigures* if they move, in a collision-free way, to a certain target configuration. A robot ensemble *gathers* if, no matter where each robot starts, there is a vertex z , such that eventually every robot is in z . A robot ensemble *collaboratively explores* a graph if, no matter where they start, every node is eventually visited by at least one robot. All of these tasks have *safe* variations: the robots complete their task without entering certain pre-designated “bad” nodes.

Multi-Robot Task Logic — MRTL. We now define MRTL, a logic for formally expressing robot tasks. We first define the syntax and semantics, and then we give some example formulas. Later (in Lemma 1) we prove that, when restricted to bounded-switching orderings, MRTL formulas (and therefore many interesting natural tasks) can be converted into MSOL formulas over graphs.

MRTL Syntax. Fix $k \in \mathbb{N}$ and Σ . Formulas of MRTL_k are built, as in the definition of $\text{MSOL}(\Sigma)$ from Section 2, from the following atomic formulas: $x = y$, edg_σ (for $\sigma \in \Sigma$), $x \in X$, true , and the following additional atomic formulas (with free variables $\bar{X}, \bar{x}, \bar{y}$ each of size k) $\text{Reach}_Q, \text{Halt}_Q^K, \text{Infty}_Q$ and Rept_Q^K where $Q \in \{\exists, \forall\}$ and $\emptyset \neq K \subseteq [k]$. Denote by MRTL the set of formulas $\cup_k \text{MRTL}_k$.

MRTL Semantics. Formulas of MRTL_k are interpreted over graphs G , and with respect to k -robot ensembles \bar{R} and a set of orderings Ω . Define the satisfaction relation $\models_{\bar{R}, \Omega}$:

- $G \models_{\bar{R}, \Omega} \text{Reach}_\exists(\bar{X}, \bar{x}, \bar{y})$ iff there is an ordering $\alpha \in \Omega$ and there is a finite run of \bar{R} on G that uses a schedule that follows α , such that the run starts with some initial configuration of the form $\langle \bar{x}, \bar{p} \rangle$ (i.e., $\bar{p} \in \prod I_i$), ends with a configuration of the form $\langle \bar{y}, \bar{q} \rangle$ (i.e., $\bar{q} \in \prod Q_i$), and for each $i \in [k]$, the set of positions of robot i on this run is contained in X_i .
- $G \models_{\bar{R}, \Omega} \text{Halt}_\exists^K(\bar{X}, \bar{x}, \bar{y})$ means the same as Reach_\exists except that the last tuple of states \bar{q} has the property that $i \in K$ implies that $q_i \in H_i$ (i.e., every robot in K is in a halting state).
- $G \models_{\bar{R}, \Omega} \text{Infty}_\exists(\bar{X}, \bar{x}, \bar{y})$ means the same as Reach_\exists except that the run is infinite and, instead of ending in \bar{y} , it visits \bar{y} infinitely often.
- $G \models_{\bar{R}, \Omega} \text{Rept}_\exists^K(\bar{X}, \bar{x}, \bar{y})$ means the same as Reach_\exists except that the run is infinite, and infinitely often it reaches a configuration of the form $\langle \bar{y}, \bar{q} \rangle$ such that $i \in K$ implies that q_i is an accepting state (i.e., $q_i \in A_i$).
- $G \models_{\bar{R}, \Omega} \text{Reach}_\forall(\bar{X}, \bar{x}, \bar{y})$ is the same as Reach_\exists except replace “there is an ordering $\alpha \in \Omega$ and there is a finite run...” by “for every ordering $\alpha \in \Omega$ there is a finite run ...”. In a similar way, define $\text{Halt}_\forall^K, \text{Infty}_\forall$ and Rept_\forall^K .

Extend the satisfaction relation to all formulas of MRTL_k in the natural way.

Example 1. The statement $G \models_{\bar{R}, \Omega} (\forall \bar{x})(\exists \bar{y})(\exists \bar{X}) \text{Reach}_\exists(\bar{X}, \bar{x}, \bar{y}) \wedge (\wedge_{i,j} y_i = y_j)$ means that, no matter where the robots start in G , there is an ordering $\alpha \in \Omega$, and a run according to a schedule that follows α , such that the robots \bar{R} gather at some vertex of the graph G . Replacing Reach_\exists by Reach_\forall means, no matter where the robots start in G , for every ordering $\alpha \in \Omega$, the robots have a run according to a schedule that follows α such that the robots gather at a vertex of the graph. Note that by conjuncting with $\wedge_i X_i \cap B = \emptyset$ where B is an MSOL -definable set of “bad” vertices, one can express “safe gathering”.

Example 2. Consider the statement $G \models_{\bar{R}, \Omega} (\forall \bar{x})(\exists \bar{y})[\text{NONLEAF}(\bar{x}) \wedge \text{DIFF}(\bar{x}) \rightarrow (\text{LEAF}(\bar{y}) \wedge \text{DIFF}(\bar{y}) \wedge \text{Reach}_\forall(V^k, \bar{x}, \bar{y}))]$ where G is a tree, $\text{NONLEAF}(\bar{x})$ is an MSOL -formula expressing that every x_i is not a leaf, $\text{LEAF}(\bar{y})$ is an MSOL -formula expressing that every y_i is a leaf, and $\text{DIFF}(\bar{z})$ is an MSOL -formula expressing that $z_i \neq z_j$ for $i \neq j$. The statement says that, as long as the robots start on different internal nodes of the tree G , for every ordering $\alpha \in \Omega$ there is a run of the robots \bar{R} according to a schedule that follows α in which the robots reconfigure and arrive at different leaves.

4 Reasoning about Robot Systems

We formalise the parameterised verification problem for robot protocols and then study its decidability. The parameterised verification problem depends on a (typically infinite) set of graphs \mathcal{G} , a set of k -robot ensembles \mathcal{R} , a k -robot task written as an MRTL_k formula T , and a set of k -orderings Ω .

Definition 1. *The parameterised verification problem $\text{PVP}_{T,\Omega}(\mathcal{G}, \mathcal{R})$ is: given a robot ensemble \bar{R} from \mathcal{R} , decide whether for every graph $G \in \mathcal{G}$, $G \models_{\bar{R}, \Omega} T$ (i.e., the robots \bar{R} achieves the task T on G with orderings restricted to Ω).*

Example 3. Let \mathcal{G} be the set of all binary trees, \mathcal{R} be the set of all k -robot ensembles, let $\Omega_b := \{\alpha \in [k]^*: ||\alpha|| = b\}$ be the set of b -switch orderings, and let T be the task expressing that if the robots start on different internal nodes of a tree then they eventually reconfigure themselves to be on different leaves of the tree, no matter which ordering from Ω_b is chosen (cf. Example 2). We will see later that one can decide $\text{PVP}_{T,\Omega_b}(\mathcal{G}, \mathcal{R})$ given $b \in \mathbb{N}$. So, one can decide, given b , whether the protocol from the reconfiguration example (in the Introduction) succeeds for every ordering with b switches.

In Section 4.1 we show that the PVP is undecidable even on lines, for simple tasks, and allowing the robots very restricted testing abilities, i.e., a robot can sense which of the other robots shares the same position with it, called “local collision tests”. In Section 4.2 we show that we can guarantee decidability merely by restricting the scheduling regime while allowing the robots full testing abilities, including testing positions and states of other robots “remotely”.

4.1 Undecidability of Multi-Robot Systems on a Line

Our undecidability proof proceeds by reducing the halting problem of two counter machines to the parameterised verification problem. An *input-free 2-counter machine* (2CM) [28] is a deterministic program manipulating two nonnegative integer counters using commands that can increment a counter by 1, decrement a counter by 1, and check whether a counter is equal to zero. We refer to the “line numbers” of the program code as the “states” of the machine. One of these states is called the *halting state*, and once it is entered the machine *halts*. Observe that a 2CM has a single computation, and that if it halts then the values of both counters are bounded by some integer n . The *non-halting problem for 2CMs* is to decide, given a 2CM \mathfrak{M} , whether it does not halt. This problem is known to be undecidable [28], and is usually a convenient choice for proving undecidability of problems concerning parameterised systems due to the simplicity of the operations of counter machines.

Let \mathcal{G} be the set of all graphs that are finite lines. Formally, for every $n \in \mathbb{N}$ there is a graph $L_n = (V_n, E_n, \Sigma, \lambda_n) \in \mathcal{G}$, where $\Sigma = \{l, r\}$, $V_n = [n]$, $E_n = \cup_{i < n} \{(i, i+1), (i+1, i)\}$, and the label λ_n of an edge of the form $(i, i+1)$ is r , and of the form $(i+1, i)$ is l . We now describe how, given a 2CM \mathfrak{M} , one

can construct a robot ensemble \bar{R} which can, on long enough lines, simulate the computation of \mathfrak{M} . Our robots have very limited sensing abilities: a robot can only sense if it at one of the two ends of the line or not, and it can sense which of the other robots are in the same node as it is (“collision detection”). Note that a robot does not know that another robot has collided with it (and then moved on) if it is not scheduled while they both occupy the same node.

The basic encoding uses two counter robots C_1 and C_2 . The current position of C_i on the line corresponds to the current value of counter i , and it moves to the right to increment counter i and to the left to decrement it. Each of these robots also stores in its finite memory the current state of the 2CM. One difficulty with this basic encoding is how to ensure that the two counter robots always stay synchronised in the sense that they both agree on the next command to simulate, i.e., we need to prevent one of them from “running ahead”. A second difficulty is how to update the state of the 2CM stored by a counter robot when it simulates a command that is a test for zero of the other counter. Note that both of these difficulties are very easy to overcome if one robot can remotely sense the state/position of the other robot. Since we disallow such powerful sensing these difficulties become substantially harder to overcome. The basic idea used to overcome the first difficulty is to add synchronisation robots and have a counter robot move only if it has collided with the appropriate synchronization robot. Thus, by arranging that the synchronization robots collide with the counter robots in a round-robin way the latter alternate their simulation turns and are kept coordinated. In order to enforce this round-robin behavior we have to change the encoding such that only every other position on the line is used to encode the counter values. Thus, an increment or a decrement is simulated by a counter robot moving two steps (instead of one) in the correct direction. The basic ingredient in addressing the second difficulty is to add a *zero-test* robot that, whenever one counter is zero, moves to the position of the other counter’s robot, thus signaling to it that the first counter is zero.

Theorem 1. *For every 2CM \mathfrak{M} , there is a robot ensemble \bar{R} which, for every $n \geq 5$, simulates on the line L_n any prefix of the computation of \mathfrak{M} in which the counters never exceed the value $(n - 3)/2$.*

Proof. The ensemble \bar{R} consists of 9 robots: the *counter* robots C_0, C_1 , four *synchronisation* robots R_0, R_1, R_2, R_3 , a *zero-test* robot T , a *zero* robot Z that marks the zero position of the counters, and a *mover* robot M whose role is to ensure that the robots can simulate more than one command of \mathfrak{M} only if their starting positions on the line are as in the *initialised configuration* escribed below ((‡)). The value of a counter is encoded as half the distance between the corresponding counter robot and the Z robot (e.g., if Z is in node 3 and C_1 is in node 7 then the value of counter 1 is 2).

((‡)) (*initialised configuration*): robots R_2, R_3 are in node 1, robots R_0, R_1 are in node 2, and the rest are in node 3.

The definition of the transitions of the robots has the important property that there is only one possible run starting from the initialised configuration,

i.e., at each point in time exactly one robot has exactly one transition with a test that evaluates to *true*. We assume that each robot remembers if it is at an odd or even node. This can be done even without looking at the node by storing the parity of the number of steps taken since the initialised configuration (\ddagger).

Each command of the 2CM \mathfrak{M} is simulated by the ensemble using 4 phases. For every $i \in \{0, 1, 2, 3\}$, phase i has the following internal stages: (1) the synchronization robots arrange themselves to signal to robot R_i that it can start moving to the right (this mechanism is described below (\star)). (2) robot R_i moves to the right until it collides with robot C_j (where $j = i \bmod 2$). It is an invariant of the run that this collision is at an even node if i is odd, and vice-versa. (3) robot C_j moves one step to the left or to the right, in order to simulate the relevant half of the current command of \mathfrak{M} , as described below (\dagger) . (4) robot R_i moves to the right until it reaches the end of the line. Observe that if during this stage R_i collides with C_j then (unlike in stage 2) it is on a node with the same parity as i (by the invariant, and since C_j moved one step in stage 3). This parity information is used by C_j to know that it should not move, and by R_i to know that it can continue moving to the right. (5) robot R_i moves to the left until it reaches the beginning of the line (see (\star)), which ends the phase (here, as in the previous stage, the parity information is used to ignore collisions with C_j). In case the other counter (i.e., counter $1 - j$) is zero, stage (2) of phases 0, 1 are modified as follows: when robot R_i enters node 3 from the left it collides with Z, C_{1-j} and T ; then, T and R_i move to the right together, where T always goes first, and then R_i follows in lock-step; at the end of stage 2 both T and R_i collide with C_j , thus signalling to the latter that counter $1 - j$ is zero. A similar modification to stages (4) and (5) makes T and R_i move in lock-step fashion all the way to the right and then back to the left depositing T back in node 3.

(\dagger) : The operation performed by C_j in stage (3) of each phase is as follows. In phase 0 robot C_0 simulates the first half of the command, in phase 1 robot C_1 simulates the first half of the same command, in phase 2 robot C_0 simulates the second half of the command and in phase 3 C_1 does so. For example, if the command is “increment counter 0” then in phase 0 robot C_0 moves right one step (and updates its simulated state of \mathfrak{M} to be the next command of \mathfrak{M}), in phase 1 robot C_1 moves right one step (and updates its simulated state of \mathfrak{M}), in phase 2 robot C_0 moves again one step to the right (thus encoding an incremented counter), and in phase 3 robot C_0 moves left one step (thus, returning to its previously encoded value). Simulating the other three increment and decrement commands is done similarly. The only other command we need to simulate is of the form ”if counter i is zero goto state p else goto state q ”. Since this command does not change the value of any counter it is simulated by each counter robot going right in the first half of the simulation and left in the second half. The internal state of \mathfrak{M} is updated to p or q depending on the value of the counter. When simulating the first half of the command, robot C_j knows that counter j (resp. $1 - j$) is zero iff it sees Z (resp. T) with it.

(\star) : We now show that every arrangement of the synchronization robots uniquely determines which one of them its turn it is to move. Let $\text{NEXT}(i) :=$

$i + 1 \bmod 4$, $\text{PREV}(i) := i - 1 \bmod 4$. An *initial arrangement for phase i* is of the following form: $R_{\text{PREV}(\text{PREV}(i))}, R_{\text{PREV}(i)}$ are in node 1, and $R_i, R_{\text{next}(i)}$ are in node 2. Note that the initialised configuration (\ddagger) contains the initial arrangement for phase 0. We let the initial arrangement for phase i signal that the next robot to move is $R_{\text{PREV}(\text{PREV}(i))}$, which moves to the right, thus completing stage (1) of phase i . Hence, at the beginning of stage 2 the arrangement is such that only $R_{\text{PREV}(i)}$ is left in node 1, which signals that R_i is the next robot to move, as needed for stage (2). Just before the end of stage (5), robot R_i returns to node 2 from the left, and the above arrangement repeats itself. Hence, again it is R_i that moves, however, this time to the left (as indicated by its now different internal memory). The resulting arrangement at the end of phase i is thus: $R_{\text{prev}(i)}, R_i$ are in node 1 and $R_{\text{NEXT}(i)}, R_{\text{PREV}(\text{PREV}(i))}$ are in node 2. Observe that this is exactly the initial arrangement for phase $\text{NEXT}(i)$, as required. Note that since robots have collision tests a robot can tell by sensing which other robots are with it (and which are not) exactly which arrangement of the ones described above it is in, and thus if it is allowed to move or not.

Finally, we describe how to amend the construction above by incorporating the robot M to ensure that robots can only simulate the 2CM if they happen to begin in the initialised configuration (\ddagger), and otherwise the system deadlocks after a few steps without any robot entering a halting state.⁶ Add to every transition of robot R_i , for $i \in \{0, 1, 2, 3\}$, the additional guard that M is on the same node with it. Thus, M enables the synchronisation robots to move, and if M ever stops, then so does the simulation. Robot M behaves as follows. It first verifies that the rest of the robots are in the initialised configuration by executing the following sequence (and stopping forever if any of the conditions in the sequence fail to hold): check that it is alone on the right-hand side of the line, move left until it collides with C_0, C_1, Z, T , move one step left and check that it collides with R_0, R_1 , move one step left and check it is on the left-hand side of the line and collides with R_2, R_3 . Once it verified that the robots are on the nodes specified by (\ddagger), it starts “chasing after” the currently active synchronisation robot, i.e., it remembers which robot is active and the direction it moves in, and moves in that direction (if it does not currently collide with that robot). \square

From the previous theorem we can easily deduce that \mathfrak{M} halts iff there is a run of the ensemble \overline{R} (on a long enough line, and that fully simulates the run of \mathfrak{M}) and in which the robots C_0, C_1 halt. We thus get:

Corollary 1. *Let $k = 9$, \mathcal{G} be the set of lines, \mathcal{R} the set of k -robot ensembles consisting of robots whose only tests are local collision tests and the ability to test the left (resp. right) end of the line, Ω the set of all k -orderings, and T the MRTL formula $(\forall \bar{x})(\forall \bar{y})(\forall \bar{X})\neg Halt_{\exists}^{\{1\}}(\bar{X}, \bar{x}, \bar{y})$.⁷ Then $\text{PVP}_{\text{T}, \Omega}(\mathcal{G}, \mathcal{R})$ is undecidable.*

⁶ One can modify the construction to remove the need for the M robot, however we find the exposition with M clearer.

⁷ The formula expresses “for every initial configuration, and every scheduling of the robots, robot 1 never enters a halting state”.

Suitable changes to the construction in Theorem 1 yield that other tasks, such as “certain robots gather” or “certain robots reconfigure”, are also undecidable.

Remark 1. Note that in the construction, starting from the initialised configuration, at most one robot can move at any time. Thus, allowing all robots that can act to act, as in the synchronous model, does not change anything. So, with minor modifications to deal with the initialisation phase, the theorem also holds for the synchronous model. This strengthens the previously known fact that the PVP is undecidable for synchronous robots on a line with remote testing abilities (i.e., robot l can test if “robots i and j are in the same node”) [30].

4.2 Decidability of Multi-Robot Systems with Bounded Switching

The previous section shows that decidability cannot be achieved in very limited situations. However, we now suggest a limitation on the *orderings* which guarantees decidability without requiring any other restrictions. Thus it works on many classes of graphs, robots, and tasks. We first describe, at a high level, the approach we use to solve (restricted cases of) the parameterised verification problem $\text{PVP}_{T,\Omega}(\mathcal{G}, \mathcal{R})$, cf. [30]. Suppose we can build, for every k -ensemble \bar{R} of robots, a formula $\phi_{\bar{R}, T, \Omega}$ such that for all graphs $G \in \mathcal{G}$ the following are equivalent: i) $G \models \phi_{\bar{R}, T, \Omega}$ and ii) \bar{R} achieves task T on G with orderings restricted to Ω . Then, for every \mathcal{R} and \mathcal{G} , we would have reduced the parameterised verification problem $\text{PVP}_{T,\Omega}(\mathcal{G}, \mathcal{R})$ to the $\Phi_{\mathcal{R}, T, \Omega}$ -validity problem for \mathcal{G} where $\Phi_{\mathcal{R}, T, \Omega}$ is the set of formulas $\{\phi_{\bar{R}, T, \Omega} : \bar{R} \in \mathcal{R}\}$. We now show how to build an MSOL-formula $\phi_{\bar{R}, T, \Omega}$ in case T is a formula of MRTL and Ω is a finite set of finite orderings.

We begin with a lemma that will be used as a building block. In the simplest setting, the lemma says that for every $i \in [k]$ there is an MSOL formula with free variables \bar{x}, \bar{y} that holds on a graph G if and only if robot i can move in G from x_i to y_i while all the other robots are frozen, i.e., $x_j = y_j$ for $j \neq i$.

Lemma 1 (From Robots to MSOL). *Fix k , and let \bar{R} be a k -robot ensemble over instruction set $\text{INS}_{\Sigma,k}$. For every $\bar{p}, \bar{q} \in \prod Q_i$ (k -tuples of states) and ordering $\alpha \in [k]^+$, one can effectively construct an MSOL(Σ) formula $\psi_{\alpha, \bar{p}, \bar{q}}(\bar{X}, \bar{x}, \bar{y})$ with free variables X_i, x_i, y_i ($i \in [k]$) such that for every graph G : $G \models \psi_{\alpha, \bar{p}, \bar{q}}(\bar{X}, \bar{x}, \bar{y})$ if and only if there exists a run c of \bar{R} on G according to a schedule that follows α , starting from configuration $c_1 = \langle \bar{x}, \bar{p} \rangle$ and reaching, for some $T \in \mathbb{N}$, the configuration $c_T = \langle \bar{y}, \bar{q} \rangle$, such that for all $i \in [k]$, the set of positions of robot i on $c_1 c_2 \dots c_T$ is contained in X_i .*

Similarly one can construct $\psi_{\alpha, \bar{p}, \bar{q}}^\infty(\bar{X}, \bar{x}, \bar{y})$ so that for every graph G : $G \models \psi_{\alpha, \bar{p}, \bar{q}}^\infty(\bar{X}, \bar{x}, \bar{y})$ if and only if there exists a run c of \bar{R} on G according to a schedule that follows α , starting from configuration $c_1 = \langle \bar{x}, \bar{p} \rangle$ and reaching the configuration $\langle \bar{y}, \bar{q} \rangle$ infinitely often, and such that the set of positions of robot i on the run is contained in X_i ($i \in [k]$).

Proof. Fix k and \bar{R} . We start with an auxiliary step. For $i \in [k]$, states $p_i, q_i \in Q_i$, and $\bar{s} = (s_1, \dots, s_{i-1}, s_{i+1}, \dots, s_k)$ with $s_j \in Q_j$, we define an MSO-formula

$\phi_{i,p_i,q_i,\bar{s}}$ with free variables X, x, y, \bar{z} where $\bar{z} = (z_1, \dots, z_{i-1}, z_{i+1}, \dots, z_k)$ such that $G \models \phi_{i,p_i,q_i,\bar{s}}$ if and only if the k -ensemble robot \bar{R} has a run according to a schedule in i^* in which robot i starts in position x and state p_i , and reaches position y and state q_i while only visiting vertices in X , and for $j \neq i$, robot j is in vertex z_j and state s_j and does not move or change state. This is done as follows. Each robot $R_i = \langle Q_i, \delta_i \rangle$ is a finite automaton (without initial or final states) over the finite alphabet $\text{INS}_{\Sigma,k}$. By Kleene's theorem, we can build a regular expression \exp_i (that depends on p_i, q_i, \bar{s}) over $\text{INS}_{\Sigma,k}$ for the language of the automaton R_i with initial state p_i and final state q_i . By induction on the regular expressions we build MSOL formulas (with free variables X, x, y, \bar{z}):

- $\varphi_\emptyset := \text{false}$ and $\varphi_\epsilon := x = y \wedge x \in X$,
- if $\tau \in \text{MSOL}_k(\Sigma)$ is a position-test then
 - $\varphi_{\tau \rightarrow \uparrow_\sigma} := \tau(z_1, \dots, z_{i-1}, x, z_{i+1}, \dots, z_k) \wedge \text{edg}_\sigma(x, y) \wedge x, y \in X$,
 - $\varphi_{\tau \rightarrow \circlearrowleft} := \tau(z_1, \dots, z_{i-1}, x, z_{i+1}, \dots, z_k) \wedge x = y \wedge x \in X$,
- if τ is a state-test, say “robot j is in state l ” (for $j \neq i$ and $l \in Q_j$) then, if $s_j = l$ then
 - $\varphi_{\tau \rightarrow \uparrow_\sigma} := \text{edg}_\sigma(x, y) \wedge x, y \in X$,
 - $\varphi_{\tau \rightarrow \circlearrowleft} := x = y \wedge x \in X$,
 and otherwise if $s_j \neq l$, then $\varphi_{\tau \rightarrow \uparrow_\sigma}$ and $\varphi_{\tau \rightarrow \circlearrowleft}$ are defined to be false ,
- $\varphi_{r+s} := \varphi_r \vee \varphi_s$,
- $\varphi_{r \cdot s} := \exists w [\varphi_r(X, x, w, \bar{z}) \wedge \varphi_s(X, w, y, \bar{z})]$,
- $\varphi_{r^*} := \forall Z [(cl_{\varphi_r}(X, Z, \bar{z}) \wedge x \in Z) \rightarrow y \in Z]$ where $cl_{\varphi_r}(X, Z, \bar{z})$ is defined as $\forall a, b [(a \in Z \wedge \varphi_r(X, a, b, \bar{z})) \rightarrow b \in Z]$.

Then, define $\phi_{i,p_i,q_i,\bar{s}}$ to be $\varphi_{\exp_i}(X, x, y, \bar{z})$. To prove the lemma proceed by induction on the length l of α . Base case: For $\alpha = i \in [k]$, define $\psi_{i,\bar{p},\bar{q}}(\bar{X}, \bar{x}, \bar{y})$ by $\bigwedge_{j \neq i} x_j = y_j \wedge X_j = \{x_j\} \wedge \phi_{i,p_i,q_i,\bar{s}}(X_i, x_i, y_i, x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_k)$, where $\bar{s} = (p_1, \dots, p_{i-1}, p_{i+1}, \dots, p_k)$, if $q_j = p_j$ for all $j \neq i$, and otherwise the formula is defined as false . Inductive case: For $\alpha \in [k]^+, i \in [k]$, define $\psi_{\alpha \cdot i, \bar{p}, \bar{q}}(\bar{X}, \bar{x}, \bar{y})$ by $\exists \bar{z} \bigvee_{\bar{r}} [\psi_{\alpha, \bar{p}, \bar{r}}(\bar{X}, \bar{x}, \bar{z}) \wedge \psi_{i, \bar{r}, \bar{q}}(\bar{X}, \bar{z}, \bar{y})]$ and \bar{r} varies over $\prod Q_i$. This completes the construction of ψ_α . The construction of ψ_α^∞ is similar. \square

In Lemma 1, a variable X_i designates a set containing – but not necessarily equal to – the positions of robot i along the run. If one wishes X_i to designate the exact set of positions visited by robot i (in order to express, e.g., “exploration”), then one needs to modify the construction of $\phi_{i,p_i,q_i,\bar{s}}$ in the proof of the lemma.⁸ The required modifications are straightforward except for those to the definition of φ_{r^*} , which are more complicated.⁹

⁸ In [30] it is wrongly stated that one can transform an MSOL formula that says that there is a run (satisfying some property) that stays within a set X , to one that says that it also visits all of X , by simply requiring that X be a minimal set for which a run satisfying the property exists.

⁹ Recall that φ_{r^*} has free variables X, x, y, \bar{z} , and its semantic in this case is that robot i can reach y from x (with the other robots' positions being \bar{z}), visiting exactly X , using a concatenation of sub-paths each satisfying φ_r . Intuitively, φ_{r^*} existentially quantifies over the stitching points of these sub-paths and uses appropriate sub-formulas that are all satisfied iff one can find sub-paths that can be stitched to lead from x to y and that cover all the positions in X .

Observe that this lemma can be used to express both collaborative and adversarial scheduling. For instance, if Ω is a finite set of orderings, the formula $\bigvee_{\alpha \in \Omega} \psi_{\alpha, \bar{p}, \bar{q}}(\bar{X}, \bar{x}, \bar{y})$ says that there is an ordering $\alpha \in \Omega$ that the robots can follow to go from \bar{x} to \bar{y} while staying in \bar{X} , i.e., the ordering is chosen collaboratively, while $\bigwedge_{\alpha \in \Omega} \psi_{\alpha, \bar{p}, \bar{q}}(\bar{X}, \bar{x}, \bar{y})$ expresses that the ordering is chosen adversarially.

Putting everything together, we solve the PVP for finite sets of orderings (and thus for adversarial or co-operative b -switch orderings $\Omega_b := \{\alpha : \|\alpha\| = b\}$).

Theorem 2. *There is an algorithm that given an edge-label set Σ , a number of robots $k \in \mathbb{N}$, a formula T of MRTL $_k$, a finite set Ω of finite k -orderings, and a description of a context-free set of Σ -graphs \mathcal{G} , decides PVP $_{T, \Omega}(\mathcal{G}, \mathcal{R})$, where \mathcal{R} is the set of all k -ensembles of robots over $\text{INS}_{\Sigma, k}$.*

Proof. Given $\bar{R} \in \mathcal{R}$ build the formula $\phi_{\bar{R}, T, \Omega}$ by replacing every atomic formula in T by its definition with respect to \bar{R} . E.g., $\text{Reach}_{\exists}(\bar{X}, \bar{x}, \bar{y})$ is replaced by $\bigvee_{\alpha \in \Omega} \bigvee_{\bar{p}} \bigvee_{\bar{q}} \psi_{\alpha, \bar{p}, \bar{q}}(\bar{X}, \bar{x}, \bar{y})$, where \bar{p} varies over $\prod_{i \in [k]} I_i$ and \bar{q} varies over $\prod_{i \in [k]} Q_i$. Now, a routine induction on the structure of the formula T shows that $G \models_{\bar{R}, \Omega} T$ if and only if $G \models \phi_{\bar{R}, T, \Omega}$. By Lemma 1 the formula $\phi_{\bar{R}, T, \Omega}$ is in MSOL(Σ). Finally, apply the fact that the MSOL-validity problem for context-free sets of graphs \mathcal{G} is uniformly decidable [9]. \square

5 Discussion

In [6, 30] (see also the discussion before Theorem 1) it was shown that the PVP is undecidable for two synchronous robots on a line, reachability tasks, and allowing the robots “remote” position-tests. In Section 4.1 we substantially strengthen this result and prove that the problem is still undecidable even if we only allow robots “local” position-tests or even just local “collision tests”, both for robots that move synchronously and asynchronously. The fact that the proof works for both the synchronous and asynchronous models (Remark 1), strongly suggests that limiting the robots’ sensing capabilities may not be a very fruitful direction for decidability. In Section 4.2 we showed that for asynchronous robots, if one imposes a bound on the number of times the robots can switch, then PVP is decidable for very general tasks (i.e., those expressible in a new logic called MRTL), large classes of graphs (i.e., the context-free sets of graphs), and allowing robots very powerful testing abilities (i.e., MSOL position-tests and state-tests). This is the first parameterised decidability result of the PVP for *multiple* robots where the environment is the parameter. Thus, our work indicates that if practitioners want formal guarantees on the correctness of the robot protocols they design, then they could design them in the framework given in this paper (i.e., finite-state, bounded-switching, powerful testing abilities).

A main limitation of our decidability result is the fact that the set of grids is not context-free — grids are the canonical workspaces since they abstract 2D and 3D real-world scenarios. However, this limitation is inherent and not

confined to our formalisation since the parameterised verification problem even for one robot ($k = 1$) on a grid with only “local” tests is undecidable [6, 30]. A second limitation is that robots do not have a rich memory (e.g., they cannot remember a map of where they have visited). Extending the abilities to allow for richer memory and communication will result in undecidability, unless it is done in a careful way. Also, the complexity of the decision procedure we gave is very high. Again this is inherent in the problem since, e.g., already for one robot on trees the PVP with the “explore and halt” task is EXPTIME-complete [30]. We leave for future research the problem of finding decidability results with reasonable complexity for multi-robot systems that are rich enough to capture protocols found in the distributed computing literature, e.g., [5, 24, 14, 15].

References

1. B. Aminof, S. Jacobs, A. Khalimov, and S. Rubin. Parameterized model checking of token-passing systems. In *VMCAI*, volume 8318 of *LNCS*, pages 262–281. Springer, 2014.
2. B. Aminof, T. Kotek, F. Spegni, S. Rubin, and H. Veith. Parameterized model checking of rendezvous systems. In *CONCUR*, volume 8704 of *LNCS*, pages 109–124. Springer, 2014.
3. B. Aminof, S. Rubin, F. Zuleger, and F. Spegni. Liveness of parameterized timed networks. In M. M. Halldórsson, K. Iwama, N. Kobayashi, and B. Speckmann, editors, *ICALP*, volume 9135 of *LNCS*, pages 375–387. Springer, 2015.
4. C. Auger, Z. Bouzid, P. Courtieu, S. Tixeuil, and X. Urbain. Certified impossibility results for byzantine-tolerant mobile robots. In *SSS*, volume 8255 of *LNCS*, pages 178–190. Springer, 2013.
5. M. A. Bender and D. K. Slonim. The power of team exploration: Two robots can learn unlabeled directed graphs. Technical report, MIT, 1995.
6. M. Blum and C. Hewitt. Automata on a 2-dimensional tape. *SWAT (FOCS)*, pages 155–160, 1967.
7. P. Čermák, A. Lomuscio, F. Mogavero, and A. Murano. Mcmas-slk: A model checker for the verification of strategy logic specifications. In *CAV*, volume 8559 of *LNCS*, pages 525–532. Springer, 2014.
8. R. Cohen, P. Fraigniaud, D. Ilcinkas, A. Korman, and D. Peleg. Label-guided graph exploration by a finite automaton. *T. on Algorithms (TALG)*, 4(4):42, 2008.
9. B. Courcelle and J. Engelfriet. Book: Graph structure and monadic second-order logic. a language-theoretic approach. *Bull. EATCS*, 108:179, 2012.
10. S. Das. Mobile agents in distributed computing: Network exploration. *Bull. EATCS*, 109:54–69, 2013.
11. G. De Giacomo, P. Felli, F. Patrizi, and S. Sardiña. Two-player game structures for generalized planning and agent composition. In M. Fox and D. Poole, editors, *AAAI*, pages 297–302, 2010.
12. G. Delzanno. Parameterized verification and model checking for distributed broadcast protocols. In *ICGT*, volume 8571 of *LNCS*, pages 1–16. Springer, 2014.
13. K. Diks, P. Fraigniaud, E. Kranakis, and A. Pelc. Tree exploration with little memory. *Journal of Algorithms*, 51(1):38–63, 2004.
14. P. Flocchini, G. Prencipe, and N. Santoro. Computing by mobile robotic sensors. In S. Nikoletseas and J. D. Rolim, editors, *Theoretical Aspects of Distributed Computing in Sensor Networks*, EATCS, pages 655–693. Springer, 2011.

15. P. Flocchini, G. Prencipe, and N. Santoro. *Distributed Computing by Oblivious Mobile Robots*. Synthesis Lectures on Distributed Computing Theory. Morgan & Claypool, 2012.
16. P. Flocchini, G. Prencipe, N. Santoro, and P. Widmayer. Hard tasks for weak robots: The role of common knowledge in pattern formation by autonomous mobile robots. In *Algorithms and Computation*, volume 1741 of *LNCS*, pages 93–102. Springer, 1999.
17. P. Fraigniaud, D. Ilcinkas, G. Peer, A. Pelc, and D. Peleg. Graph exploration by a finite automaton. *Theoretical Computer Science*, 345:331 – 344, 2005.
18. L. Gasieniec and T. Radzik. Memory efficient anonymous graph exploration. In H. Broersma, T. Erlebach, T. Friedetzky, and D. Paulusma, editors, *Graph-Theoretic Concepts in Computer Science*, volume 5344 of *LNCS*, pages 14–29. Springer, 2008.
19. Y. Hu and G. De Giacomo. Generalized planning: Synthesizing plans that work for multiple environments. In T. Walsh, editor, *IJCAI*, pages 918–923. AAAI, 2011.
20. A. Khalimov, S. Jacobs, and R. Bloem. PARTY parameterized synthesis of token rings. In N. Sharygina and H. Veith, editors, *CAV*, volume 8044 of *LNCS*, pages 928–933. Springer, 2013.
21. A. Khalimov, S. Jacobs, and R. Bloem. Towards efficient parameterized synthesis. In R. Giacobazzi, J. Berdine, and I. Mastroeni, editors, *VMCAI*, volume 7737 of *LNCS*, pages 108–127. Springer, 2013.
22. P. Kouvaros and A. Lomuscio. Automatic verification of parameterised multi-agent systems. In M. L. Gini, O. Shehory, T. Ito, and C. M. Jonker, editors, *AAMAS*, pages 861–868, 2013.
23. P. Kouvaros and A. Lomuscio. A counter abstraction technique for the verification of robot swarms. In B. Bonet and S. Koenig, editors, *AAAI*, pages 2081–2088, 2015.
24. E. Kranakis, D. Krizanc, and S. Rajsbaum. Mobile agent rendezvous: A survey. In P. Flocchini and L. Gasieniec, editors, *SIROCCO*, volume 4056 of *LNCS*, pages 1–9. Springer, 2006.
25. E. Kranakis, D. Krizanc, and S. Rajsbaum. Computing with mobile agents in distributed networks. In S. Rajasekaran and J. Reif, editors, *Handbook of Parallel Computing: Models, Algorithms, and Applications*, CRC Computer and Information Science Series, pages 8–1 to 8–20. Chapman Hall, 2007.
26. N. A. Lynch. *Distributed Algorithms*. Morgan Kaufmann, 1996.
27. L. Millet, M. Potop-Butucaru, N. Sznajder, and S. Tixeuil. On the synthesis of mobile robots algorithms: The case of ring gathering. In P. Felber and V. K. Garg, editors, *SSS*, volume 8756 of *LNCS*, pages 237–251. Springer, 2014.
28. M. L. Minsky. *Computation: finite and infinite machines*. Prentice-Hall, Inc., 1967.
29. A. Murano and L. Sorrentino. A game-based model for human-robots interaction. In *Workshop "From Objects to Agents" (WOA)*, volume 1382 of *CEUR Workshop Proceedings*, pages 146–150. CEUR-WS.org, 2015.
30. S. Rubin. Parameterised verification of autonomous mobile-agents in static but unknown environments. In G. Weiss, P. Yolum, R. H. Bordini, and E. Elkind, editors, *AAMAS*, pages 199–208, 2015.
31. I. Suzuki. Proving properties of a ring of finite-state machines. *Inf. Process. Lett.*, 28(4):213–214, July 1988.

Liveness of Parameterized Timed Networks ^{*}

Benjamin Aminof¹, Sasha Rubin², Florian Zuleger¹, and Francesco Sogno³

¹ TU Vienna, Austria

² Università degli Studi di Napoli “Federico II”, Italy

³ Università Politecnica delle Marche, Ancona, Italy

Abstract. We consider the model checking problem of infinite state systems given in the form of parameterized discrete timed networks with multiple clocks. We show that this problem is decidable with respect to specifications given by B- or S-automata. Such specifications are very expressive (they strictly subsume ω -regular specifications), and easily express complex liveness and safety properties. Our results are obtained by modeling the passage of time using symmetric broadcast, and by solving the model checking problem of parameterized systems of untimed processes communicating using k -wise rendezvous and symmetric broadcast. Our decidability proof makes use of automata theory, rational linear programming, and geometric reasoning for solving certain reachability questions in vector addition systems; we believe these proof techniques will be useful in solving related problems.

1 Introduction

Timed automata — finite state automata enriched by a finite number of dense- or discrete-valued clocks — can be used to model more realistic circuits and protocols than untimed systems [3,7]. A timed network consists of an arbitrary but fixed number of timed automata running in parallel [2,1]. In each computation step, either some fixed number of automata synchronize by a rendezvous-transition or time advances. We consider the *parameterized model-checking problem (PMCP)* for timed networks: Does a given specification (usually given by a suitable automaton) hold *for every system size*? Apart from a single result which deals with much weaker synchronization than rendezvous [13], no positive PMCP results for liveness specifications of timed automata are known.

System model: In this paper we prove the decidability of the PMCP for discrete timed networks with no controller and liveness specifications. To do this, we reduce the PMCP of these timed networks to the PMCP of *RB-systems* — systems of finite automata communicating via *k -wise rendezvous* and *symmetric*

^{*} Benjamin Aminof and Florian Zuleger were supported by the Austrian National Research Network S11403-N23 (RiSE) of the Austrian Science Fund (FWF) and by the Vienna Science and Technology Fund (WWTF) through grant ICT12-059. Sasha Rubin is a Marie Curie fellow of the Istituto Nazionale di Alta Matematica.

broadcast. This broadcast action is symmetric in the sense that there is no designated sender. In contrast, the standard broadcast action can distinguish between sender and receivers, and so the PMCP of liveness properties is undecidable even in the untimed setting [9].

Our Techniques and Results: Classical automata (e.g., nondeterministic Büchi word automata (NBW)) are not able to capture the behaviors of RB-systems. Thus, our decidability result uses nondeterministic BS-automata (and their fragments B- and S-automata) which strictly subsume NBW [6].

We show that the PMCP is decidable for controllerless discrete timed networks and (and systems communicating via k -wise rendezvous and symmetric broadcast) and specifications given by B-automata or S-automata (and in particular by NBW) or for negative specifications (i.e., the set of bad executions) given by BS-automata. We prove decidability by constructing a B-automaton that precisely characterizes the runs of a timed network from the point of view of a single process. Along the way, we also obtain an EXPSPACE upper bound for the PMCP of safety properties of discrete timed networks.

In order to build the B-automaton, an intricate analysis of the interaction between the transitions caused by the passage of time (modeled by broadcasts) which involve all processes, and those that are the result of rendezvousing processes, is needed. It is this interaction that makes the problem complicated. Thus, for example, results concerning pairwise rendezvous without broadcast [11] do not extend to our case. Our solution to this problem involves the introduction of the idea of a *rational relaxation* of a Vector Addition System, and geometric lemmas concerning paths in these relaxations. It is important to note that these vector addition systems can not capture the edges that correspond to the passage of time. However, they provide the much needed flexibility in capturing what happens in between time ticks *in the presence of these ticks*.

Related Work. Discrete timed networks with rendezvous and a controller were introduced in [1] where it was shown that safety is decidable using the technique of well-structured transition systems. Their result implies a non-elementary upper bound (which we improve to EXPSPACE) for the complexity of the PMCP of safety properties of timed networks without a controller. PMCP of liveness properties for continuous-time networks with a controller process is undecidable [2]. However, their proof heavily relies on time being dense and on the availability of a distinguished controller process. RB-systems with a controller were introduced in [12] where it is proved that under an additional strong restriction on the environment and process templates (called a shared simulation), such systems admit cutoffs that allow one to model check epistemic-temporal logic of the parameterised systems. The main difference between our work and theirs is: we do not have a controller, we make no additional restrictions, and we can model check specifications given by B- or S-automata. The authors in [13] proved that the PMCP is decidable for continuous timed networks synchronizing using conjunctive Boolean guards and MTL and TCTL specifications. Finally, there are many decidability and undecidability results in the untimed setting, e.g.,[14,9,8,4,5].

2 Definitions and Preliminaries

Labeled Transition Systems. A *(edge-)labeled transition system (LTS)* is a tuple $\langle S, I, R, \Sigma \rangle$, where S is the set of *states* (usually $S \subseteq \mathbb{N}$), $I \subseteq S$ are the *initial states*, $R \subseteq S \times \Sigma \times S$ is the *edge relation*, and Σ is the *edge-labels alphabet*. *Paths* are sequences of transitions, and *runs* are paths starting in initial states.

Automata. We use standard notation and results of automata, such as nondeterministic Büchi word automata (NBW) [15]. A *BS-word automaton (BSW)* ([6]) is a tuple $\langle \Sigma, Q, Q_0, \Gamma, \delta, \Phi \rangle$ where Σ is a finite *input alphabet*, Q is a set of *states*, $Q_0 \subseteq Q$ is a set of *initial states*, Γ is a set of *counter (names)*, $\delta \subseteq Q \times \Sigma \times \mathcal{C}^* \times Q$ is the *transition relation* where \mathcal{C} is the set of *counter operations*, i.e. $c := 0, c := c + 1, c := d$ for $c, d \in \Gamma$, and Φ is the *acceptance condition* described below. A run ρ is defined like for nondeterministic automata over infinite words by ignoring the \mathcal{C}^* component. Denote by $c(\rho, i)$ the i th value assumed by counter $c \in \Gamma$ along ρ . The acceptance condition Φ is a positive Boolean combination of the following conditions ($q \in Q, c \in \Gamma$): (i) q is visited infinitely often (Büchi-condition); (ii) $\limsup_i c(\rho, i) < \infty$ (B-condition); (iii) $\liminf_i c(\rho, i) = \infty$ (S-condition). An automaton that does not use B-conditions is called an *S-automaton (SW)*, and one that does not use S-conditions is called a *B-automaton (BW)*.

It is known that BSWs are relatively well behaved [6]: their emptiness problem is decidable; they are closed under union and intersection, but not complement; and BW (resp. SW) can be complemented to SW (resp. BW). Since BSWs are not closed under complement, we are forced, if we are to use the automata-theoretic approach for model checking (cf. [15]), to give the specification in terms of the undesired behaviours, or to consider specifications in terms of BWs or SWs (which both strictly extend ω -regular languages).

Rendezvous with Symmetric Broadcast (RB-System). Intuitively, RB-systems describe the parallel composition of $n \in \mathbb{N}$ copies of a process *template*. An RB-system evolves nondeterministically: either a k -wise rendezvous action is taken, i.e., k different processes instantaneously synchronize on a rendezvous action \mathbf{a} , or the symmetric broadcast action is taken, i.e., all processes must take an edge labeled by \mathbf{b} . Systems without the broadcast action are called R-systems.

In the rest of the paper, fix k (the number of processes participating in a rendezvous), a finite set Σ_{actn} of *rendezvous actions*, the *rendezvous alphabet* $\Sigma_{\text{rdz}} = \cup_{\mathbf{a} \in \Sigma_{\text{actn}}} \{\mathbf{a}_1, \dots, \mathbf{a}_k\}$, and the *communication alphabet* Σ_{com} which is the union $\{(i_1, \mathbf{a}_1), \dots, (i_k, \mathbf{a}_k)\} \mid \mathbf{a} \in \Sigma_{\text{actn}}, i_j \in \mathbb{N}, j \in [k]\} \cup \{\mathbf{b}\}$.

A *process template* (or *RB-template*) is a finite LTS $P = \langle S, I, R, \Sigma_{\text{rdz}} \cup \{\mathbf{b}\} \rangle$ such that for every state $s \in S$ there is a transition $(s, \mathbf{b}, s') \in R$ for some $s' \in S$. We call edges labeled by \mathbf{b} *broadcast edges*, and the rest *rendezvous edges*. For ease of exposition, we assume (with one notable exception, namely P^\sim defined in Section 3) that for every $\varsigma \in \Sigma_{\text{rdz}}$ there is at most one edge in P labeled by ς

and we denote it by $\text{edge}(\varsigma)$.⁴ . The *RB-system* \mathcal{P}^n is defined, given a template P and $n \in \mathbb{N}$, as the finite LTS $\langle Q^n, Q_0^n, \Delta^n, \Sigma_{\text{com}} \rangle$ ⁵ where:

1. Q^n is the set of functions (called *configurations*) of the form $f : [n] \rightarrow S$. We call $f(i)$ the *state of process i* in f . Note that we sometimes find it convenient to consider a more flexible naming of processes in which we let Q^n be the set of functions $f : X \rightarrow S$, where $X \subset \mathbb{N}$ is some set of size n .
2. The set of *initial configurations* $Q_0^n = \{f \in Q^n \mid f(i) \in I \text{ for all } i \in [n]\}$ consists of all configurations which map all processes to initial states of P .
3. The set of *global transitions* Δ^n are tuples $(f, \sigma, g) \in Q^n \times \Sigma_{\text{com}} \times Q^n$ where one of the following two conditions hold:
 - $\sigma = \mathbf{b}$, and for every $i \in [n]$ we have that $(f(i), \mathbf{b}, g(i)) \in R$. This is called a *broadcast transition*.
 - $\sigma = ((i_1, \mathbf{a}_1), \dots, (i_k, \mathbf{a}_k))$, where $\mathbf{a} \in \Sigma_{\text{actn}}$ is the *action* taken, and $\{i_1, \dots, i_k\} \subseteq [n]$ are k different processes. In this case, for every $1 \leq j \leq k$ we have that $(f(i_j), \mathbf{a}_j, g(i_j)) \in R$; and $f(i) = g(i)$ for every $i \notin \{i_1, \dots, i_k\}$. This is called a *rendezvous transition*, and the processes in the set $\text{prcs}(\sigma) := \{i_1, \dots, i_k\}$ are called the *rendezvousing processes*.

We denote the *action taken* on a global transition $t = (f, \sigma, g)$ by $\text{actn}(t)$. Thus, $\text{actn}(t) := \mathbf{a}$ if $\sigma = ((i_1, \mathbf{a}_1), \dots, (i_k, \mathbf{a}_k))$, and otherwise $\text{actn}(t) := \mathbf{b}$.

A process template P induces the *infinite RB-system* \mathcal{P} , i.e., the LTS $\mathcal{P} = \langle Q, Q_0, \Delta, \Sigma_{\text{com}} \rangle$ where $Q = \cup_{n \in \mathbb{N}} Q^n$, $Q_0 = \cup_{n \in \mathbb{N}} Q_0^n$, $\Delta = \cup_{n \in \mathbb{N}} \Delta^n$.

Executions of an RB-System, and the Parameterized Model-Checking Problem. Given a global transition $t = (f, \sigma, g)$, and a process i , we say that i *moved* in t iff: $\sigma = \mathbf{b}$, or $i \in \text{prcs}(\sigma)$. We write $\text{edge}_i(t)$ for the edge of P taken by process i in the transition t , and \perp if i did not move in t . Thus, if $\sigma = \mathbf{b}$ then $\text{edge}_i(t) := (f(i), \mathbf{b}, g(i))$; and if $\sigma = ((i_1, \mathbf{a}_1), \dots, (i_k, \mathbf{a}_k))$ then $\text{edge}_i(t) := (f(i), \mathbf{a}_j, g(i))$ if $\sigma(j) = (i, \mathbf{a}_j)$ for some $j \in [k]$, and otherwise $\text{edge}_i(t) := \perp$. Take an RB-System $\mathcal{P}^n = \langle Q^n, Q_0^n, \Delta^n, \Sigma_{\text{com}} \rangle$, a path $\pi = t_1 t_2 \dots$ in \mathcal{P}^n , and a process i in \mathcal{P}^n . Define $\text{proj}_\pi(i) := \text{edge}_i(t_{j_1}) \text{edge}_i(t_{j_2}) \dots$, where $j_1 < j_2 < \dots$ are all the indices j for which $\text{edge}_i(t_j) \neq \perp$. Intuitively, $\text{proj}_\pi(i)$ is the path in P taken by process i during the path π . Define the set of *executions* $\text{EXEC}_{\mathcal{P}}$ of \mathcal{P} to be the set of the runs of \mathcal{P} projected onto a single process. Note that, due to symmetry, we can assume w.l.o.g. that the runs are projected onto process 1. Formally, $\text{EXEC}_{\mathcal{P}} = \{\text{proj}_\pi(1) \mid \pi \text{ is a run of } \mathcal{P}\}$. We denote by $\text{EXEC}_{\mathcal{P}}^{\text{fin}}$ (resp. $\text{EXEC}_{\mathcal{P}}^\infty$) the finite (infinite) executions in $\text{EXEC}_{\mathcal{P}}$.

For specifications \mathcal{F} (e.g., LTL, NFWs) interpreted over infinite (resp. finite) words over the alphabet $S \times (\Sigma_{\text{rdz}} \cup \{\mathbf{b}\}) \times S$ of transitions,⁶ the *Parameterized Model Checking Problem* (PMCP) for \mathcal{F} is to decide, given a template P , and a specification $\varphi \in \mathcal{F}$, if all executions in $\text{EXEC}_{\mathcal{P}}^\infty$ (resp. $\text{EXEC}_{\mathcal{P}}^{\text{fin}}$) satisfy φ .

⁴ This can always be assumed by increasing the size of the rendezvous alphabet.

⁵ Even though Σ_{com} is infinite, Δ^n refers only to a finite subset of it.

⁶ In this way we can also capture atomic propositions on edges or states since these atoms may be pushed into the rendezvous label.

Discrete Timed Networks. We refer the reader to [1] for a formal definition of timed networks. Here we describe the templates and informally describe the semantics. Fix a set C of *clocks*. A *timed network template* is a finite LTS $\langle Q, I, R, \Sigma_{\text{rdz}} \rangle$. We associate to each letter $a_i \in \Sigma_{\text{rdz}}$ a *command* $r(a_i) \subseteq C$ and a *guard* $p(a_i)$. A guard p is a Boolean combination of predicates of the form $c \bowtie x$ where $c \in \mathbb{N}$ is a constant, $x \in C$ is a clock, and $\bowtie \in \{<, =\}$.

Intuitively, a discrete timed network consists of the parallel composition of $n \in \mathbb{N}$ template processes, each running a copy of the template. Each copy has a local state (q, t) , where $q \in Q$ and $t : C \rightarrow \mathbb{N}$. A rendezvous action a is *enabled* if there are k processes in local states (q_i, t_i) ($i \in [k]$) and there are edges $(q_i, a, q'_i) \in R$ such that the clocks t_i satisfy the guards $p(a_i)$. The rendezvous action is *taken* means that the k processes change state (to q'_i) and each of the clocks in $r(a_i)$ is reset to 0. The network evolves non-deterministically, in steps: either all clocks advance by one time unit (so every $t(c)$ increases by one)⁷ or a rendezvous action $a \in \Sigma_{\text{rdz}}$ is taken. For a timed network template T let \mathcal{T}^n denote the timed network composed of $n \in \mathbb{N}$ templates T and let \mathcal{T} denote the union of the networks \mathcal{T}^n for $n \in \mathbb{N}$.

Given a timed network template T one can build an equivalent RB-template P , i.e., $\text{EXEC}_P = \text{EXEC}_{\mathcal{T}}$. The key insight is that the passage of time, that causes all clocks to advance by one time unit, is simulated by symmetric broadcast, and timed-guards are pushed into the template states. The RB-system P requires only a finite number of states since clock values bigger than the greatest constant appearing on the guards are collapsed to a single abstract value (cf. [1]).

Useful lemmas. We state a few simple but useful lemmas. The first lemma states that, by partitioning processes of an RB-system into independent groups, a system with many processes can simulate in a single run multiple runs of smaller systems. If the simulated paths contain no broadcasts then the transitions of the simulated paths can be interleaved in any order. Otherwise, all simulated runs must have the same number of broadcasts, and the simulations of all the edges before the i 'th broadcast on each simulated path must complete before taking the i 'th broadcast on the simulating combined path.

Lemma 1 (RB-System Composition). *A system P^n can, using a single run, partition its processes into groups each simulating a run of a smaller system. All simulated paths must have the same number of broadcasts.*

Consider now an RB-system P^n , and two configurations f, f' in it such that the number of processes in each state in f is equal to that in f' , i.e., such that $|f^{-1}(s)| = |f'^{-1}(s)|$ for every $s \in S$. We call f, f' *twins*. A finite path π of length m for which $\text{src}(\pi_1)$ and $\text{dst}(\pi_m)$ are twins is called a *pseudo-cycle*. For example, for P in Figure 1, the following path in P^4 is a pseudo-cycle that is not a cycle: $(p, q, q, r) \xrightarrow{((3,c_1),(4,c_2))} (p, q, r, p) \xrightarrow{((2,c_1),(3,c_2))} (p, r, p, p) \xrightarrow{((3,a_1),(4,a_2))} (p, r, q, q)$.

Lemma 2. *By renaming processes after each iteration, a pseudo-cycle π can be pumped to an infinite path which repeatedly goes through the actions on π .*

⁷ Alternatively, as in [1], one can let time advance by any amount.

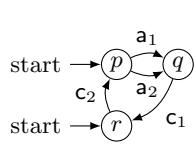


Fig. 1: R-template with $k = 2$.

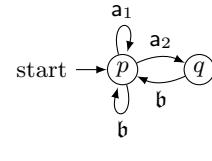


Fig. 2: RB-template with $k = 2$.

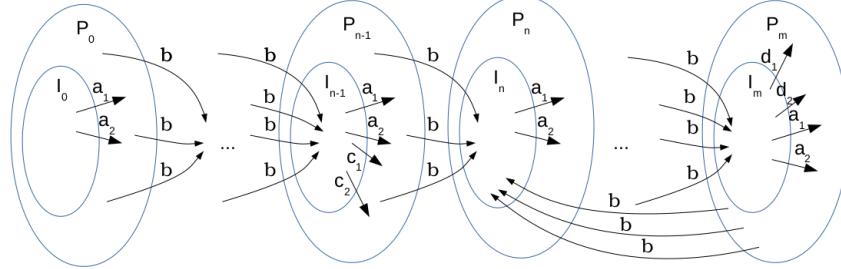


Fig. 3: A high level view of the reachability-unwinding lasso.

3 The Reachability-Unwinding of a Process Template

Given template $P = \langle S, I, R, \Sigma_{\text{rdz}} \cup \{\mathbf{b}\} \rangle$, our goal in this section is to construct a new process template $P^{\circ} = \langle S^{\circ}, I^{\circ}, R^{\circ}, \Sigma_{\text{rdz}} \cup \{\mathbf{b}\} \rangle$, called the *reachability-unwinding* of P , see Figure 3. The template P° will play a role in all our algorithms for solving the PMCP of RB-systems. Intuitively, P° is obtained by alternating the following two operations: (i) taking a copy of P and removing from it all unreachable rendezvous edges; and (ii) unwinding on broadcast edges. This is repeated until a copy is created which is equal to a previous one, we then stop and close the unwinding back into the old copy, forming a high-level lasso structure.

Technically, it is more convenient to first calculate all the desired copies and then to arrange them in the lasso. Thus, we first calculate, for $0 \leq i \leq m$ (for an appropriate m), an R-template $P_i = \langle S_i, I_i, R_i, \Sigma_{\text{rdz}} \rangle$ which is a copy of P with initial states redesignated and all broadcast edges, plus some rendezvous edges, removed. Second, we take P_0, \dots, P_m and combine them, to create the single process template P° , by connecting the states in P_i with the initial states of P_{i+1} (P_n for $i = m$, where $n \leq m$ is determined by the lasso structure) with broadcast edges, as naturally induced by P .

Construct the R-template $P_i = \langle S_i, I_i, R_i, \Sigma_{\text{rdz}} \rangle$ (called the i 'th component of P°) recursively: for $i = 0$, we let $I_0 := I$; and for $i > 0$ we let $I_i := \{s \in S \mid (h, \mathbf{b}, s) \in R \text{ for some } h \in S_{i-1}\}$ be the set of states reachable from S_{i-1} by a broadcast edge. The elements S_i and R_i are obtained using the following *saturation* algorithm, which is essentially a breadth-first search: start with $S_i := I_i$ and $R_i := \emptyset$; at each round of the algorithm, consider in turn each edge $e = (s, a_h, t) \in R \setminus R_i$; if for every $l \in [k]$ there is some edge $(s', a_l, t') \in R$ with $s' \in S_i$, then add e to R_i and add t (if not already there) to S_i . The algorithm ends when

a fixed-point is reached. Observe a property of this algorithm: if $(s, \mathbf{a}_h, t) \in R_i$ then for all $l \in [k] \setminus \{h\}$ there exists $s', t' \in S_i$ such that $(s', \mathbf{a}_l, t') \in R_i$.

Now, P_i is completely determined by I_i (and P), and so there are at most $2^{|S|}$ possible values for it. Hence, for some $n \leq m < 2^{|S|}$ it must be that $P_n = P_{m+1}$. We stop calculating P_i 's when this happens since for every $i \in \mathbb{N}_0$ it must be that $P_i = P_{n+((i-n) \bmod r)}$, where $r = m+1-n$. We call n the *prefix length* of P° (usually denoted by ψ), call r the *period* of P° , and for $i \in \mathbb{N}_0$, call $n + ((i-n) \bmod r)$ the *associated component number* of i , and denote it by $\text{comp}(i)$.

We now construct from P_0, \dots, P_m the template $P^\circ = \langle S^\circ, I^\circ, R^\circ, \Sigma_{\text{rdz}} \cup \{\mathbf{b}\} \rangle$, as follows: (i) $S^\circ := \bigcup_{i=0}^m (S_i \times \{i\})$; (ii) $I^\circ := I_0 \times \{0\}$ (recall that we also have $I_0 = I$); (iii) R° contains the following transitions: the rendezvous transitions $\bigcup_{i=0}^m \{((s, i), \varsigma, (t, i)) \mid (s, \varsigma, t) \in R_i\}$, and the broadcast transitions $\bigcup_{i=0}^{m-1} \{((s, i), \mathbf{b}, (t, i+1)) \mid (s, \mathbf{b}, t) \in R \text{ and } s \in S_i\}$ and $\{((s, m), \mathbf{b}, (t, n)) \mid (s, \mathbf{b}, t) \in R \text{ and } s \in S_m\}$.

We will abuse notation, and talk about the component P_i , referring sometimes to P_i as defined before (i.e., without the annotation with i), and sometimes to the part of P° that was obtained by annotating the elements of P_i with i .

Observe that, by projecting out the component numbers (we will denote this projecting by superscript \odot) from states in P° (i.e., by replacing $(s, i) \in S^\circ$ with $s \in S$), states and transitions in P° induce states and transitions in P . Similarly, paths and runs in P° can be turned into paths and runs in P . We claim that also the converse is true, i.e., that by adding component numbers, states and transitions in P can be lifted to ones in P° ; and that by adding the correct (i.e., reflecting the number of previous broadcasts) component numbers to the states of the transitions of a run in P , it too can be lifted to a run in P° . However, a path in P that is not a run (i.e., that does not start at an initial configuration), may not always be lifted to a path in P° due to the removal of unreachable edges in the components making up P° .

The next lemma says that we may work with template P° instead of P .

Lemma 3. *For every $n \in \mathbb{N}$, we have that $\text{runs}(\mathcal{P}^n) = \{\rho^\odot \mid \rho \in \text{runs}((\mathcal{P}^\circ)^n)\}$.*

The following lemma says, intuitively, that for every component P_i there is a run of P° that “loads” arbitrarily many processes into every state of P_i .

Lemma 4. *For all $b, n \in \mathbb{N}$ there is a finite run π of P° with b broadcasts, s.t., $|f^{-1}(s)| \geq n$ for all states s in the component $P_{\text{comp}(b)}$, where $f = \text{dst}(\pi)$. \square*

The following lemma states that the set of finite executions of the RB-system \mathcal{P} is equal to the set of finite runs of the process template P° (modulo component numbers). This is very convenient since, whereas \mathcal{P} is infinite, P° is finite. Unfortunately, when it comes to infinite executions of \mathcal{P} we only get that they are contained in (though in many cases not equal to) the set of infinite runs of P° . This last observation is also true for P : consider for example Figure 2 without the \mathbf{b} edges, and an infinite repetition of the self loop.

Lemma 5. $\text{EXEC}_{\mathcal{P}}^{\text{fin}} = \{\pi^\odot \mid \pi \in \text{runs}(P^\circ), |\pi| \in \mathbb{N}\}$; and $\text{EXEC}_{\mathcal{P}}^\infty \subseteq \{\pi^\odot \mid \pi \in \text{runs}(P^\circ), |\pi| = \infty\}$

Solving PMCP for regular specifications. Given $P = \langle S, I, R, \Sigma_{\text{rdz}} \cup \{\mathbf{b}\} \rangle$, let $\mathcal{A}_P^{\text{fin}}$ denote the reachability-unwinding P^\sim viewed as an automaton (NFW), with all states being accepting states, and transitions e are labeled e^\circledcirc (i.e., they have the component number removed). Formally, $\mathcal{A}_P^{\text{fin}} = \langle R, S^\sim, I^\sim, R', S^\sim \rangle$, so the input alphabet of $\mathcal{A}_P^{\text{fin}}$ is R (the transition relation of P), and $R' := \{(s, (s^\circledcirc, \sigma, t^\circledcirc), t) \mid (s, \sigma, t) \in R^\sim\} \subseteq S^\sim \times R \times S^\sim$. Hence:

Theorem 1. *The PMCP of RB-systems (resp. discrete timed networks) for regular specifications is in PSPACE (resp. EXPSPACE)*

4 Solving PMCP of Liveness Specifications

In this section we show how to solve the PMCP for specifications concerning infinite executions. We begin with the following lemma showing that, if we want to use the automata theoretic approach, classical automata models (e.g. Büchi, Parity) are not up to the task.

Lemma 6. *There is a process template P such that EXEC_P^∞ is not ω -regular.*

Proof. Consider the process template given in Figure 2. It is not hard to see that in every infinite run of P^n there may be at most $n - 1$ consecutive rendezvous transitions before a broadcast transition, resetting all processes to state 1, is taken. Overall, we have that EXEC_P^∞ is the set of words of the form $a_1^{n_1} a_2^{m_1} \mathbf{b} a_1^{n_2} a_2^{m_2} \mathbf{b} \dots$, where $m_i \in \{0, 1\}$ for every i , and $\limsup n_i < \infty$. This language is not ω -regular since the intersection of its complement with $\{a_1, \mathbf{b}\}^\omega$ is not ω -regular (because it contains no ultimately periodic words). \square

In light of Lemma 6, we turn our attention to a stronger model, called BSW [6]. Thus, we solve the PMCP for liveness specifications as follows: given a process template P , we show how to build a BSW \mathcal{A}_P^∞ accepting exactly the executions in EXEC_P^∞ . Model checking of a specification given by a BSW \mathcal{A}' accepting all undesired (i.e., bad) executions, is thus reduced to checking for the emptiness of the intersection of \mathcal{A}_P^∞ and \mathcal{A}' .

Defining the Automaton \mathcal{A}_P^∞ . We now describe the structure of the BSW \mathcal{A}_P^∞ (in fact we define a BW) accepting exactly the executions in EXEC_P^∞ .

An important element in the construction is a classification of the edges in P^\sim into four types: blue, green, orange, and red. The red edges are those that appear at most finitely many times on any execution in EXEC_P^∞ . An edge is blue if it appears infinitely many times on some execution in EXEC_P^∞ with finitely many broadcasts, but only finitely many times on every execution which has infinitely many broadcasts. An edge e is green if there is some run $\pi \in \text{EXEC}_P^\infty$ with infinitely many broadcasts on which e appears unboundedly many times between broadcasts, i.e., if for every $n \in \mathbb{N}$ there are $i < j \in \mathbb{N}$ such that $\pi_i \dots \pi_j$ contains n occurrences of e and no broadcast edges. An edge which is neither blue, green, nor red is orange. By definition, blue and green edges are not broadcast edges. Since the set EXEC_P^∞ is infinite, it is not at all clear that the

problem of determining the type of an edge is decidable. Indeed, this turns out to be a complicated question, and we dedicate Section 4.1 to show that one can decide the type of an edge.

The automaton $\mathcal{A}_{\mathcal{P}}^{\infty}$ is made up of three copies of $\mathcal{A}_{\mathcal{P}}^{fin}$ (called $\mathcal{A}_{\mathcal{P}}^{\infty 1}$, $\mathcal{A}_{\mathcal{P}}^{\infty 2}$, $\mathcal{A}_{\mathcal{P}}^{\infty 3}$), as follows: $\mathcal{A}_{\mathcal{P}}^{\infty 1}$ is an exact copy of $\mathcal{A}_{\mathcal{P}}^{fin}$; the copy $\mathcal{A}_{\mathcal{P}}^{\infty 2}$ has only the green and orange edges left; and $\mathcal{A}_{\mathcal{P}}^{\infty 3}$ has only the blue and green edges left (and in particular has no broadcast edges). Furthermore, for every edge (s, σ, s') in $\mathcal{A}_{\mathcal{P}}^{\infty 1}$ we add two new edges, both with the same source as the original edge, but one going to the copy of s' in the copy $\mathcal{A}_{\mathcal{P}}^{\infty 2}$, and one to the copy of s' in the copy $\mathcal{A}_{\mathcal{P}}^{\infty 3}$. The initial states of $\mathcal{A}_{\mathcal{P}}^{\infty}$ are the initial states of $\mathcal{A}_{\mathcal{P}}^{\infty 1}$. For the acceptance condition: every state in $\mathcal{A}_{\mathcal{P}}^{\infty 2}$ and $\mathcal{A}_{\mathcal{P}}^{\infty 3}$ is a Büchi-state, and there is a single counter $C \in \Gamma_B$ that is incremented whenever an orange rendezvous edge is taken in $\mathcal{A}_{\mathcal{P}}^{\infty 2}$ and reset if a broadcast edge is taken in $\mathcal{A}_{\mathcal{P}}^{\infty 2}$.

Formally, given a process template $P = \langle S, I, R, \Sigma_{rdz} \cup \{\mathbf{b}\} \rangle$ and its unwinding $P^{\infty} = \langle S^{\infty}, I^{\infty}, R^{\infty}, \Sigma_{rdz} \cup \{\mathbf{b}\} \rangle$ define $\mathcal{A}_{\mathcal{P}}^{\infty} = \langle \Sigma, Q, Q_0, \Gamma, \delta, \Phi \rangle$ as:

- The input alphabet Σ is the edge relation R of template P .
- The state set Q is $\{(i, s) \mid s \in S^{\infty}, i \in \{1, 2, 3\}\}$.
- The initial state set Q_0 is $\{(1, s) \mid s \in I^{\infty}\}$.
- There is one counter, $\Gamma = \{c\}$.
- The transition relation δ is $\delta_1 \cup \delta_2 \cup \delta_3$, where: δ_1 consists of all tuples $((1, s_1), (s_1^{\odot}, \sigma, s_2^{\odot}), \epsilon, (i, s_2))$ such that $(s_1, \sigma, s_2) \in R^{\infty}, i \in \{1, 2, 3\}$; and δ_3 consists of all tuples $((3, s_1), (s_1^{\odot}, \sigma, s_2^{\odot}), \epsilon, (3, s_2))$ such that $(s_1, \sigma, s_2) \in R^{\infty}$ is blue or green; and δ_2 consists of all tuples $((2, s_1), (s_1^{\odot}, \sigma, s_2^{\odot}), upd^{\sigma, \rho}, (2, s_2))$ such that $\rho := (s_1, \sigma, s_2) \in R^{\infty}$ is green or orange, and $upd^{\sigma, \rho}$ is the single operation $c := 0$ if ρ is orange and $actn(\sigma) = \mathbf{b}$, and $upd^{\sigma, \rho}$ is the single operation $c := c + 1$ if ρ is orange and $actn(\sigma) \neq \mathbf{b}$, and $upd^{\sigma, \rho}$ is the empty sequence ϵ if ρ is green. Here ϵ is the empty sequence of operations (i.e., do nothing to the counter).
- The acceptance condition Φ states that $\limsup_i c(\rho, i) < \infty$ (i.e., counter c must be bounded) and some state $q \in Q \setminus \{(1, s) \mid s \in S^{\infty}\}$ is visited infinitely often.

Lemma 7. *An edge (s_1, σ, s_2) of P^{∞} is: (i) red iff it does not appear on any pseudo-cycle of P^{∞} ; (ii) blue iff it appears on a pseudo-cycle of P^{∞} with no broadcasts, but not on any that contain broadcasts; (iii) green iff it appears on a pseudo-cycle of P^{∞} with no broadcasts, that is part of a bigger pseudo-cycle with broadcasts; (iv) orange iff it appears on a pseudo-cycle C of P^{∞} that has broadcasts, but not on any without broadcasts.*

The following lemma states that we can assume that pseudo-cycles mentioned in Lemma 7 (that have broadcasts) are of a specific form.

Lemma 8. *An edge e appears on a pseudo-cycle D in P^{∞} , which contains broadcasts, iff it appears on a pseudo cycle C of P^{∞} containing exactly r broadcast transitions and with all processes starting in the component P_n , where n, r are the prefix length and period of P^{∞} , respectively. Furthermore, C preserves any nested pseudo-cycles of D that contain no broadcasts.*

Theorem 2. *The language recognized by $\mathcal{A}_{\mathcal{P}}^\infty$ is exactly $\text{EXEC}_{\mathcal{P}}^\infty$.*

Proof (sketch). The fact that every word in $\text{EXEC}_{\mathcal{P}}^\infty$ is accepted by $\mathcal{A}_{\mathcal{P}}^\infty$ follows in a straightforward way from its construction. For the reverse direction, given $\alpha \in \text{EXEC}_{\mathcal{P}}^\infty$ with an accepting run Ω in $\mathcal{A}_{\mathcal{P}}^\infty$, we need to construct a run π in \mathcal{P} whose projection on process 1 is α . We consider the interesting case that α has infinitely many broadcasts (and thus finitely many red and blue edges). The challenging part is how to make process 1 trace the suffix β of α containing only green and orange edges. Since Ω is accepting, counter C_2 is bounded on Ω . Hence, there is a bound m on the number of orange edges in β between any r broadcasts, where r is the period of P° .

For every green (resp. orange) edge e of P that appears on β , by Lemmas 7, 8, there is a pseudo-cycle C_e with r broadcasts on which e appears. Furthermore, if e is green it actually appears on an inner pseudo-cycle of C_e without broadcasts. Let E_{green} (resp. E_{orange}) be the set of green (resp. orange) edges that appear infinitely often on α . By taking exactly enough processes to assign them to one copy of C_e for every $e \in E_{\text{green}}$, and m copies of C_e for every $e \in E_{\text{orange}}$, and composing them using Lemma 1 we can simulate all these copies of these pseudo-cycles in one pseudo-cycle D also with r broadcasts. By Lemma 2, we can pump this pseudo-cycle forever. Furthermore, between broadcasts we have freedom on how to interleave the simulations. We make process 1 trace β by making it successively swap places with the right process in the group simulating a copy of the cycle C_e where e is the next edge on β to be traced (just when the group is ready to use that edge). The key observation is that once a group is used by process 1 there are two options. If it is a group corresponding to a green edge then we can make the group (after 1 leaves it) traverse the inner pseudo-cycle (the one without broadcasts) thus making it ready to serve process 1 again. If the group corresponds to an orange edge e , then it will only be reusable when the whole pseudo-cycle C_e completes (since there is no inner pseudo-cycle to use), i.e., after r broadcasts. However, since there are m groups for each such edge, and m bounds from above the number of orange edges that need to be taken by process 1 between r broadcasts. \square

As we show (Section 4.1, Theorem 4), the problem of determining the type (blue, green, orange, or red) of an edge in P° is decidable, hence, we conclude this section by stating our main theorem (the proof is now immediate).

Theorem 3. *The PMCP (of RB-systems or discrete timed networks) for BW- or SW-specifications or complements of specifications given by BSW, is decidable.*

4.1 Deciding Edge Types

Theorem 4. *Given a process template P° , the problem of determining the type (blue, green, orange, red) of an edge e in P° is decidable.*

A key observation for proving Theorem 4 is that by Lemma 7, the type of an edge can be decided by looking for witnessing pseudo-cycles C in \mathcal{P}° . Indeed,

a witness can determine if an edge is green or not. If not, another witness can determine if it is orange or not, and the last witness can separate the blue from the red. We will show an algorithm that given an edge that is not green tells us if it is orange or not. The algorithm can be modified to check for the other types of witnesses without much difficulty.

By Lemma 8, we can assume that the pseudo-cycle C we are looking for has very specific structure. Our algorithm uses linear programming, in a novel and interesting way, to detect the existence of such a pseudo-cycle C .

Counter Representation. Given a process template $P = \langle S, I, R, \Sigma_{\text{rdz}} \cup \{\mathbf{b}\} \rangle$, let $d = |S|$, and fix once and for all some ordering s_1, s_2, \dots, s_d of the states in S . We associate with every configuration f in \mathcal{P} a vector $f^\sharp := (|f^{-1}(s_1)|, \dots, |f^{-1}(s_d)|) \in \mathbb{N}_0^d$, called the *counter representation* of f . We also associate with every transition $t = (f, \sigma, g)$ the vector $t^\sharp := g^\sharp - f^\sharp$ representing the change in the number of processes in each state. If t is a rendezvous transition then $g^\sharp - f^\sharp$ is completely determined by the action $\mathbf{a} \in \Sigma_{\text{actn}}$ taken in σ . Indeed, if $\sigma = ((i_1, \mathbf{a}_1), \dots, (i_k, \mathbf{a}_k))$ then $g^\sharp - f^\sharp = \mathbf{a}^\sharp$, where $\mathbf{a}^\sharp \in \mathbb{N}_0^d$ is the vector defined by letting $\mathbf{a}^\sharp(s) := |\{j \in [k] \mid \text{dst}(\text{edge}(a_j)) = s\}| - |\{j \in [k] \mid \text{src}(\text{edge}(a_j)) = s\}|$ for every $s \in S$.

Given $u \in \mathbb{Q}^d$, and a sequence of vectors $\varrho = \varrho_1 \dots \varrho_m$ in \mathbb{Q}^d , the pair $\rho = (u, \varrho)$ is called a path from u to $v = u + \sum_{i=1}^m \varrho_i$. We write ρ_j for the vector $u + \sum_{i=1}^j \varrho_i$, for every $0 \leq j \leq m$. The path ρ is *legal* if $\rho_j \in \mathbb{Q}_{\geq 0}^d$ for every $0 \leq j \leq m$, i.e., if no coordinate goes negative at any point. Given a finite path $\pi_1 \dots \pi_m$ in \mathcal{P} , we call the path $\pi^\sharp := (\text{src}(\pi_1)^\sharp, \pi_1^\sharp \dots \pi_m^\sharp)$ in \mathbb{Q}^d its *counter representation*. Observe that π^\sharp is always a legal path.

Rational Relaxation of VASs. Vector Addition Systems (VASs) or equivalently Petri nets are one of the most popular formal methods for the representation and the analysis of parallel processes [10]. Unfortunately, RB-systems **cannot** be modelled by VASs since a transition in a VAS only moves a constant number of processes, whereas a broadcast in an RB-system may move any number of processes. On the other hand, R-System can be modelled by VASs, and we do use this fact to analyze the behaviour of the counter representation between broadcasts. Moreover, we note that integer linear programming is a natural fit for describing paths and cycles in the counter representation. However, in order to apply linear programming to RB-systems we have to overcome two intertwined obstacles: (i) not every path in the counter representation induces a path in \mathcal{P} , and (ii) since we have no bound on the length of the pseudo-cycle C we cannot have variables describing each configuration on it, and we need to aggregate information. These obstacles are aggravated by the presence of broadcasts. Another difficulty of applying linear programming to RB-systems arises from the fact that the question of reachability in an RB-system with two (symmetric) broadcast actions and a controller is undecidable (which can be obtained by modifying a result in [9] concerning asymmetric broadcast).

The solution we propose to this problem, which we found to be surprisingly powerful, is to use linear programming but look for a solution in *rational* numbers and not in integers. Thus, we introduce the notion of the *rational relaxation* of a

VAS, obtained by allowing any non-negative rational multiple of configurations and transitions of the original VAS. Since our linear programs use homogeneous systems of equations, multiplying a rational solution by a large enough number would yield another solution in integers. Thus the scaling property obtained as a consequence of rational relaxation precludes the possibility of specifying a single controller! Thinking of the counter representation as vectors of rational numbers also allows us to use geometric reasoning to solve the two problems (i), (ii) described above. Essentially, by cutting transitions to smaller pieces (which cannot be done at will to integer vectors) and rearranging the pieces, we can transform a description of a path in an aggregated form, as it comes out of the linear program, into one which is legal and can be turned into a path in \mathcal{P} . We strongly believe that these techniques can be fruitfully used in other circumstances concerning counter-representations, and similar objects (such as vector addition systems and Petri nets).

Due to lack of space, the description of the linear programs we use, as well as the geometric machinery we develop will be published in an extended version.

References

1. Abdulla, P.A., Deneux, J., Mahata, P.: Multi-clock timed networks. In: Ganzinger, H. (ed.) LICS. pp. 345–354 (July 2004)
2. Abdulla, P.A., Jonsson, B.: Model checking of systems with many identical timed processes. TCS 290(1), 241–264 (2003)
3. Alur, R.: Timed automata. In: CAV. pp. 8–22. Springer (1999)
4. Aminof, B., Jacobs, S., Khalimov, A., Rubin, S.: Parameterized model checking of token-passing systems. In: VMCAI. pp. 262–281. Springer (2014)
5. Aminof, B., Kotek, T., Rubin, S., Spegni, F., Veith, H.: Parameterized model checking of rendezvous systems. In: CONCUR, pp. 109–124. Springer (2014)
6. Bojanczyk, M.: Beyond ω -regular languages. In: STACS 2010. pp. 11–16 (2010)
7. Chevallier, R., Encenaz-Tiphene, E., Fribourg, L., Xu, W.: Timed verification of the generic architecture of a memory circuit using parametric timed automata. Formal Methods in System Design 34(1), 59–81 (2009)
8. Delzanno, G., Sangnier, A., Zavattaro, G.: Parameterized verification of ad hoc networks. In: CONCUR. LNCS, vol. 6269, pp. 313–327 (2010)
9. Esparza, J., Finkel, A., Mayr, R.: On the verification of broadcast protocols. In: LICS. pp. 352–359 (1999)
10. Esparza, J., Nielsen, M.: Decidability issues for petri nets - a survey. Bulletin of the EATCS 52, 244–262 (1994)
11. German, S.M., Sistla, A.P.: Reasoning about systems with many processes. JACM 39(3), 675–735 (1992)
12. Kouvaros, P., Lomuscio, A.: A cutoff technique for the verification of parameterised interpreted systems with parameterised environments. In: IJCAI 2013 (2013)
13. Spalazzi, L., Spegni, F.: Parameterized model-checking of timed systems with conjunctive guards. In: Verified Software: Theories, Tools and Experiments, pp. 235–251. Springer (2014)
14. Suzuki, I.: Proving properties of a ring of finite-state machines. Inf. Process. Lett. 28(4), 213–214 (1988)
15. Vardi, M.Y.: An automata-theoretic approach to linear temporal logic. In: Logics for concurrency, pp. 238–266. Springer (1996)

On the expressive power of communication primitives in parameterised systems^{*}

Benjamin Aminof¹, Sasha Rubin², and Florian Zuleger¹

¹ Technische Universität Wien, Austria

² Università degli Studi di Napoli “Federico II”, Italy

Abstract. We study foundational problems regarding the expressive power of parameterised systems. These (infinite-state) systems are composed of arbitrarily many finite-state processes that synchronise using a given communication primitive, i.e., broadcast, asynchronous rendezvous, broadcast with message loss, pairwise rendezvous, or disjunctive guards. With each communication primitive we associate the class of parameterised systems that use it. We study the relative expressive power of these classes (can systems in one class be simulated by systems in another?) and provide a complete picture with only a single question left open. Motivated by the question of separating these classes, we also study the absolute expressive power (e.g., is the set of traces of every parameterised system of a given class ω -regular?). Our work gives insight into the verification and synthesis of parameterised systems, including new decidability and undecidability results for model checking parameterised systems using broadcast with message loss and asynchronous rendezvous.

1 Introduction

Parameterised systems are composed of arbitrarily many copies of the same finite-state process. The processes in a system run independently, but are given a mechanism by which they can synchronise, e.g., in broadcast systems one process can send a message to all the other processes, while in a rendezvous system the message is received by a single process [12,17]. Examples of such systems abound in theoretical computer science (e.g., distributed algorithms [18]) and biology (e.g., cellular processes [15]).

Problem Statement. Different synchronisation mechanisms, or *communication primitives* as we call them here, yield systems with different capabilities. For instance, broadcast is at least as expressive as rendezvous since in two steps broadcast may simulate a rendezvous (I broadcast “I want to rendezvous”, and someone broadcasts the reply “I will rendezvous with you”, illustrated in Figure 5). On the other hand, intuitively, broadcast is more expressive than rendezvous (since to simulate a broadcast a process would have to rendezvous with

^{*} Benjamin Aminof and Florian Zuleger are supported by the Austrian National Research Network S11403-N23 (RiSE) of the Austrian Science Fund (FWF) and by the Vienna Science and Technology Fund (WWTF) through grant ICT12-059. Sasha Rubin is a Marie Curie fellow of the Istituto Nazionale di Alta Matematica.

all other processes before anyone made a different move). The motivation of this paper is to formalise such reasoning and make such intuitions precise.

Communication Primitives. This paper focuses on representative primitives from the literature on formal methods for parameterised systems: Broadcast (BC), like CBP message passing can model ethernet-like broadcast, GSM’s cell-broadcast, or the `notifyAll` method in Concurrent Java [5,12,20]; Asynchronous Rendezvous (AR) can model the `notify` method in Concurrent Java [5]; Broadcast with Message Loss (BCML) can model mobile ad hoc networks (MANETS) and systems that use selective broadcast with nodes that can be activated or deactivated at any time [6,7,8]; and Pairwise Rendezvous (PR), like CSP message passing, can model population protocols [4,17]. For comparison we also consider a primitive that admits cutoffs, i.e., disjunctive guards (DG) [10], a property not shared by the previous primitives.³

Executions of Parameterized Systems. We systematically compare communication primitives using the standard notion of executions from the point of view of single processes. Indeed, many papers (e.g. [5,7,6,11,12,13,9,17,2,3]) consider specifications from the point of view of single processes — important examples of such specifications are safety specifications like coverability and liveness specifications like repeated coverability and termination. Given a process P , and a communication primitive CP , let P_{CP}^n be the finite-state system composed of n copies of P that synchronise using CP (note that there is no special “controller” process). An *execution* is a (finite or infinite) sequence of labels⁴ of states of a single process in P_{CP}^n . In many applications (e.g., in parameterised verification), one needs to consider systems of all sizes. Thus, we let P_{CP}^∞ denote the infinite-state system consisting of the (disjoint) union of the systems P_{CP}^n for each $n \in \mathbb{N}$.

Relative Expressive Power. We define the natural comparison $\text{CP} \leq_{\text{IE}} \text{CP}'$ as follows: for every process P that uses CP there is a process Q that uses CP' , such that P_{CP}^∞ and $Q_{\text{CP}'}^\infty$ have the same set of infinite executions. Similarly, we write \leq_{FE} if considering only finite executions. The informal meaning of these comparisons \leq is that CP' can simulate CP , with respect to linear-time specifications. All of our simulations (except of AR by PR) have the added properties that they also hold for systems of a fixed finite size, and that they are efficiently computable. This latter fact is useful for example for model checking (MC) classes of parameterised systems with respect to linear-time specifications (over a single process), i.e., if $\text{CP} \leq \text{CP}'$ and the translation from P to Q is efficient, then MC CP -systems is (immediately) reduced to MC CP' -systems. We remark that most decidability results for MC parameterised systems are for linear-time specifications, whereas for branching-time specifications it is typically undecidable [17,2,3,11].

Absolute Expressive Power. Motivated by the problem of comparing communication primitives, we also study their absolute expressive power. That is, a communication primitive CP determines a class of languages \mathcal{L}_{CP} , i.e., the

³ A cutoff is a maximal number of processes that needs to be model checked in order to guarantee correct behaviour of any number of processes. Our results show that, indeed, having a cutoff lowers the expressive power.

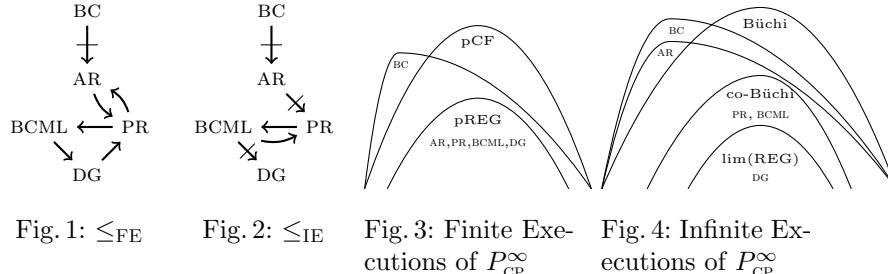
⁴ Typically, each label is a set of atomic propositions.

sets of executions of such systems. How does the class \mathcal{L}_{CP} relate to canonical classes of languages, such as regular, context-free, ω -regular, etc.? Answers to such questions allow one to deduce that certain communication primitives *cannot be simulated* by certain others, as well as directing one's choice of communication primitive for modeling and synthesis of distributed systems.

Our contributions.⁵

Relative Expressive Power. We provide a full picture of the relative expressive power of these communication primitives, see Figures 1 and 2 — an arrow from CP to CP' means $CP' \leq CP$,⁶ and a mark across an arrow means that $CP < CP'$.

Section 3 establishes all but three of the arrows in Figures 1 and 2: we get $PR \leq_{IE} BCML$ from Theorems 6 and 4, $PR \leq_{FE} DG$ from Theorems 2 and Proposition 5, and $AR \leq_{FE} PR$ from Proposition 5 and Theorem 3.



Absolute Expressive Power. The classes of languages of finite executions generated by the different primitives are illustrated in Figure 3. BC can generate languages that are not context free, whereas DG, PR, BCML, AR generate exactly the prefix-closed regular languages (pREG). However, no communication primitive can generate all prefix-closed context-free languages (pCF). The case of infinite executions is illustrated in Figure 4. We show that DG can generate exactly limits of regular languages, BCML, PR can generate exactly co-Büchi languages, AR, BC can generate non ω -regular languages, whereas no communication primitive can generate all ω -regular languages. We present our results on absolute expressive power in Section 5. The strictness of the arrows in Figures 1 and 2 follow from our results on absolute expressive power. To get $DG <_{IE} PR$ (and thus also deduce $DG <_{IE} BCML$) use Theorem 4 and Theorem 7 (and the fact that there are co-Büchi languages that are not the limit of any regular language, e.g., all words over $\{a, b\}$ with finitely many a s). To get $PR <_{IE} AR$ use Proposition 9 and Theorem 6. To get $AR <_{FE} BC$ use Proposition 6 and Theorem 3. To get $AR <_{IE} BC$ use Proposition 8 and Theorem 5.

Model Checking Linear-Time Specifications. Our techniques yield new results about model checking (MC) AR and BCML parameterised systems for liveness

⁵ For lack of space, some proofs are missing or only sketched and can be found in the full version of this paper.

⁶ We note that the transitivity of the relations \leq gives rise to additional simulations that, for clarity, are not drawn in the figures.

properties.⁷ In particular, even the simplest liveness property (i.e., does there exist an infinite run) is undecidable for AR systems (Section 4). Also, liveness properties are decidable in PTIME for systems using BCML (a problem that was not even known to be decidable); this follows because BCML can be efficiently simulated by PR (Proposition 2), and the fact that MC of PR-systems can be done in PTIME (which itself follows from [17, Section 4]).

2 Definitions and Preliminaries

Tuples f over a set X may be written (x_1, \dots, x_k) or in functional notation $f \in X^{[k]}$, i.e., $f(i) = x_i$. Given a set Σ , we denote by Σ^* , Σ^+ , Σ^ω the sets of all finite strings, all non-empty finite string, and all infinite strings, respectively, over Σ . Let u_i denote the i th letter of u . We write $\text{pre}(L)$ for the set of *finite prefixes* of some language $L \subseteq \Sigma^*$ or $L \subseteq \Sigma^\omega$. A language $L \subseteq \Sigma^*$ is *prefix closed* if $L = \text{pre}(L)$. The *limit* of a language $L \subseteq \Sigma^*$ is the language $\lim L \subseteq \Sigma^\omega$ such that $\alpha \in \lim L$ if and only if infinitely many prefixes of α are in L . A *labeled transition system (LTS)* is a tuple $\langle \Omega, A, Q, Q_0, \delta, \lambda \rangle$ where Ω is a set of *letters* (also called *observables*),⁸ A is a set of *edge labels*, Q is a finite set of *states*, $Q_0 \subseteq Q$ are the *initial states*, $\delta \subseteq Q \times A \times Q$ is the *transition relation*, and $\lambda : Q \rightarrow \Omega$ is the *labeling function*. For $\tau = (q, a, q')$ we write $\text{src}(\tau) = q$, $\text{des}(\tau) = q'$ and $\text{edglab}(\tau) = a$, and we also write $q \xrightarrow{a} q'$. A *path* of an LTS is a (finite or infinite) string of transitions $\pi := \pi_1 \pi_2 \dots$ of δ such that $\text{src}(\pi_{i+1}) = \text{des}(\pi_i)$ for every i . We write $\text{src}(\pi) := \text{src}(\pi_1)$, and if π is finite we write $\text{des}(\pi) := \text{des}(\pi_{|\pi|})$. A *run* is a path π where $\text{src}(\pi) \in Q_0$. We write $\text{edglab}(\pi)$ for the sequence $\text{edglab}(\pi_1)\text{edglab}(\pi_2)\dots$. Typically the edge-labels will carry information, i.e., an action (e.g., send message m), and whether or not the edge is visible. See [22] for basic notions about automata. In particular, we use the following acronyms: NFW, NBW and NCW where N stands for “nondeterministic”, F for “finite”, B for “Büchi”, C for “co-Büchi”, and W for “word automata”. Counter machines CM are standard variations of Minsky Machines, i.e., they have a fixed number of counters that can be incremented, decremented if not zero, and tested for zero. In the rest of this paper, the word “simulation” is used as in ordinary natural language, and not as part of the technical term “(bi)simulation relation”.

A note about simulations and visibility. In order to reason about simulations we have to be able to hide some of the inner steps involved. Consider the following motivating example. All the x86 family of processes support the same basic instruction set, but they implement each instruction using their own sequences of microcode instructions. This is fine since to the running software these sequences of microcode are invisible and it can only see their effect on the observables, i.e., the values of the registers. In order to capture this basic trait of simulations, our definition of local process labels each transition with a Boolean flag indicating whether it is visible or not, with the added condition that invisible transitions

⁷ As in [12,9] we formalise safety properties as regular sets (of finite words) and liveness properties as ω -regular sets (of infinite words).

⁸ In applications one typically takes $\Omega := 2^{\text{AP}}$ where AP is a set of atomic predicates.

do not change the observables⁹. To demonstrate, in the introduction we illustrated that BC (effectively) simulates PR by replacing every PR transition by two (successive) BC transitions. Thus, in order to preserve the set of executions, we have to hide one of these two transitions (see Figure 5).

System Model. For a set Σ , let $\Sigma_{\text{sync}} = \{\mathbf{m}!, \mathbf{m}?\mid \mathbf{m} \in \Sigma\}$ be the *synchronisation-actions*. Let Π be a set of *internal-actions*, disjoint from Σ . A *local process* is a finite LTS $P = \langle \Omega, A, S, S_0, \delta, \lambda \rangle$ where $A := (\Sigma_{\text{sync}} \cup \Pi) \times \mathbf{B}$ and for every $(q, (\sigma, b), q') \in \delta$, if $b = \mathbf{false}$ then we must have that $\lambda(q) = \lambda(q')$. A transition $\tau = (q, (\sigma, b), q')$ is called *visible* if $b = \mathbf{true}$ and *invisible* if $b = \mathbf{false}$. Thus, an invisible transition may change the state but not what is observed.

Define functions act, vis such that $\text{act}(\tau) = \sigma$ and $\text{vis}(\tau) = b$. A state $s \in S$ is *able to receive* (resp. *able to send*) message $\mathbf{m} \in \Sigma$ if there is a transition $\tau \in \delta$ with $\text{src}(\tau) = s$ and $\text{act}(\tau) = \mathbf{m}?$ (resp. $\text{act}(\tau) = \mathbf{m}!$). States and transitions of P are called *local states* and *local transitions*. Informally, local transitions with $\sigma \in \Pi$ are transitions that a single process must take alone, and are called *local internal transitions*, whereas local transitions with $\sigma \in \Sigma_{\text{sync}}$ may involve synchronising with other processes, and are called *local synchronising transitions*.

For a local process P we now define the *global system*, i.e., the composition P_{CP}^n of n -many copies of P that communicate using CP. A *global state* of P_{CP}^n is an n -tuple of elements of S , collectively S^n . For $f = (s_1, \dots, s_n), f' = (s'_1, \dots, s'_n) \in S^n$ a *global transition* $\tau = (f, \nu, f') \in S^n \times (\Sigma \cup \Pi) \times S^n$ satisfies:

1. If $\nu \in \Pi$ then there exists i and $b \in \mathbf{B}$ such that $s_i \xrightarrow{\nu, b} s'_i$, and $s_\ell = s'_\ell$ for $\ell \neq i$ (*internal transition*).
2. If $\nu = \mathbf{m} \in \Sigma$:
 - If $\text{CP} = \text{BC}$: there exist i and $b_i \in \mathbf{B}$ such that $s_i \xrightarrow{\mathbf{m}!, b_i} s'_i$ and letting R be the set of processes $j \neq i$ that are able to receive \mathbf{m} , we must have that R is non-empty and $s_j \xrightarrow{\mathbf{m}?, b_j} s'_j$ for all $j \in R$ (and some $b_j \in \mathbf{B}$), and $s_\ell = s'_\ell$ for $\ell \notin R \cup \{i\}$ (*broadcast transition*).¹⁰
 - If $\text{CP} = \text{AR}$: there exist i and $b_i \in \mathbf{B}$ such that $s_i \xrightarrow{\mathbf{m}!, b_i} s'_i$ and either: there exists $j \neq i$ such that $s_j \xrightarrow{\mathbf{m}?, b_j} s'_j$ and $s_\ell = s'_\ell$ for $\ell \neq i, j$; or there is no $j \neq i$ such that j is able to receive \mathbf{m} , and $s_\ell = s'_\ell$ for $\ell \neq i$ (*asynchronous rendezvous transition*).
 - If $\text{CP} = \text{PR}$: there exist $i \neq j$ and $b_i, b_j \in \mathbf{B}$ such that $s_i \xrightarrow{\mathbf{m}!, b_i} s'_i$ and $s_j \xrightarrow{\mathbf{m}?, b_j} s'_j$, and $s_\ell = s'_\ell$ for $\ell \neq i, j$ (*rendezvous transition*).
 - If $\text{CP} = \text{BCML}$: there exist i and $b_i \in \mathbf{B}$ such that $s_i \xrightarrow{\mathbf{m}!, b_i} s'_i$ and there is some, possibly empty, set R of processes (not containing i) such that

⁹ It is common to allow specifications (e.g., the LTL formula Gp) to be satisfied by computations that loop forever in the same state. Thus, we *don't* consider every transition in which the observables don't change to be invisible. In particular, we can have both visible and invisible self loops. Using the CPU analogy, the former corresponds to a NOP in the instruction set, and the latter to a NOP in microcode.

¹⁰ A slightly different version of BC, in which R is also allowed to be empty, also appears in the literature [12]. Our results also hold for this version.

$s_j \xrightarrow{m?_j, b_j} s'_j$ for all $j \in R$ (and some $b_j \in \mathbf{B}$), and $s_\ell = s'_\ell$ for $\ell \notin R \cup \{i\}$ (*broadcast with message loss transition*).

- If $\text{CP} = \text{DG}$: there exist $j \neq i$ and $b \in \mathbf{B}$ such that $s_i \xrightarrow{s_j?_j, b} s'_i$ and $s_j \xrightarrow{s_j!, \text{false}} s_j$, and $s_\ell = s'_\ell$ for $\ell \neq i, j$ (*guarded transition*). ¹¹

A process k is said to be *involved* in a global transition τ if it takes a local transition γ from s_k to s'_k (e.g., in all cases above process i is involved in τ). Moreover, it is *visibly involved* if $\text{vis}(\gamma) = \text{true}$.

Finally, P_{CP}^n is the LTS $\langle \Omega^n, \Sigma \cup \Pi, S^n, S_0^n, \Delta, \Lambda \rangle$ where Δ consists of the global transitions (just defined), and $\Lambda(f)(i) := \lambda(f(i))$ for every $i \in [n]$. The infinite state LTS P_{CP}^∞ is the disjoint union of P_{CP}^n for $n \in \mathbb{N}$, and it is called a *parameterised system*, or just a *system*.

Executions. Let π be a path of P_{CP}^n . We will relate π to paths in P corresponding to a single process. Fix a process index $k \in [n]$. Let $i_1 < i_2 < \dots$ be the set of indices such that process k is visibly involved in the global transition π_{i_j} , and define $s_j := \Lambda(\text{src}(\pi_{i_j}))(k) \in \Omega$ (for all j). If there are only finitely many indices $i_1 < i_2 < \dots < i_l$, we let $\text{vislet}_k(\pi)$ be the concatenation of $s_1 s_2 \dots s_l$ with the additional letter $\Lambda(\text{des}(\pi_{i_l}))(k)$ at the end. Otherwise, we set $\text{vislet}_k(\pi) := s_1 s_2 \dots$. We define the *set of 1-executions of P_{CP}^n* by

$$\text{EXEC}(P_{\text{CP}}^n) := \{ \text{vislet}_k(\pi) : k \in [n], \pi \text{ is a run of } P_{\text{CP}}^n \} \subseteq \Omega^\omega \cup \Omega^*$$

and the *set of 1-executions of P_{CP}^∞* as $\text{EXEC}(P_{\text{CP}}^\infty) := \bigcup_{n \in \mathbb{N}^+} \text{EXEC}(P_{\text{CP}}^n)$. We denote the infinite (resp. finite) elements of $\text{EXEC}(\cdot)$ by $\text{INFEXEC}(\cdot)$ (resp. $\text{FINEXEC}(\cdot)$). It is worth noting that if a run π is infinite, but $\text{vislet}_k(\pi)$ is finite, then process k was only doing finitely many meaningful moves in π (which is akin, in a system with only visible transitions, to it being scheduled only finitely many times) which is why we do not include such traces in INFEXEC .

3 Relative Expressive Power

For communication primitives CP, CP' , write $\text{CP} \leq_{\text{IE}} \text{CP}'$ if for every local process P there is a local process Q (computable from P) such that $\text{INFEXEC}(P_{\text{CP}}^\infty) = \text{INFEXEC}(Q_{\text{CP}'}^\infty)$. Similarly, define \leq_{FE} with FINEXEC replacing INFEXEC . For $x \in \{\text{IE}, \text{FE}\}$, if $\text{CP} \leq_x \text{CP}' \leq_x \text{CP}$ then write $\text{CP} \equiv_x \text{CP}'$. If $\text{CP} \leq_x \text{CP}'$ and $\text{CP} \not\equiv_x \text{CP}'$ then write $\text{CP} <_x \text{CP}'$ (and define $<_x$ similarly). Informally, if $\text{CP} \leq_x \text{CP}'$ we say that CP' *simulates* CP . Note that, in the definition of \leq_x , if there is a PTIME algorithm that given P produces the corresponding Q then we say that CP' *efficiently simulates* CP . All the simulation results $\text{CP} \leq_x \text{CP}'$ in this paper (except for $\text{AR} \leq_{\text{FE}} \text{PR}$) are efficient simulations.

¹¹ If $\text{CP} = \text{DG}$ then we also assume $\Sigma = S$ (i.e., the synchronization alphabet is the set of local states), and for every local state $s \in S$ there is a transition $s \xrightarrow{t!, a} r$ if and only if $s = r = t$ and $a = \text{false}$ (i.e., the only transition τ with $\text{act}(\tau) = s!$ is an invisible self-loop on state s).

Relationship with verification. Every regular language of finite words is called a *safety* property, and every ω -regular language of infinite words is called a *liveness* property, cf. [12]. The model checking problem for parameterised systems using CP for a given safety (resp. liveness) property L over Ω is the following: given P , decide whether or not $\text{FINEXEC}(P_{\text{CP}}^\infty) \subseteq L$ (resp. $\text{INFEXEC}(P_{\text{CP}}^\infty) \subseteq L$). This model checking problem is sometimes called the “parameterised model checking problem” or “parameterised verification”, e.g., [11]. If CP' effectively simulates CP then the parameterised verification problem for systems using CP is reducible to the parameterised verification problem for systems using CP' .

3.1 Simulations

The simulations $\text{DG} \leq_x \text{PR} \leq_x \text{BC}$, with $x \in \{\text{FE}, \text{IE}\}$, have already been discovered in the literature [9]; we illustrate $\text{PR} \leq_x \text{BC}$ in Figure 5. These results are the starting point for our fine-grained analysis. In this section we establish the simulations $\text{DG} \leq_x \text{BCML} \leq_x \text{PR} \leq_x \text{AR} \leq_x \text{BC}$ for $x \in \{\text{IE}, \text{FE}\}$. All these simulations were not previously known. In all the proofs we efficiently construct, given a local process P , a local process Q such that $\text{EXEC}(P_{\text{CP}}^n) = \text{EXEC}(Q_{\text{CP}'}^n)$.

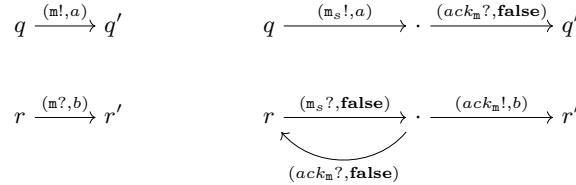


Fig. 5: Simulation of PR (left) by BC (right)

Proposition 1. $\text{AR} \leq_x \text{BC}$, for $x \in \{\text{FE}, \text{IE}\}$.

Proof. Recall that the difference between PR and AR is only that in AR a process can send a message m even if there is no other process to receive it (but if there is, then one such process must receive m). We divide the global transitions of an AR system into three types: internal transitions, synchronous transitions involving two processes, and those involving only one process. Given local process P , we build local process Q such that Q_{BC}^n simulates P_{PR}^n (for every $n > 2$), by using a sequence (called a *transaction*) of 1 or 2 global transitions. Simulating the internal transitions is done directly, the synchronous transitions involving two processes are simulated by a 2-step transaction as in the simulation of PR by BC , and the synchronous transitions in which there is only a sender are simulated by a single-step transaction as follows: let $e = (p, (m!, b), q)$ be a local transition in P ; in Q , a process can take the local transition $(p, (m_{\text{solo}}!, b), q)$ broadcasting the message that it is simulating a send of m that should have no receivers, and every process that is in a state that is able to receive m in P , receives m_{solo} and invisibly moves to a new special “disabled” copy of its current state from which it can no longer do anything; all other processes simply receive m_{solo} and invisibly

self-loop. The intuition is that by sending \mathbf{m}_{solo} process i guessed that there is no process able to receive \mathbf{m} in the simulated system, and thus we disable the processes that witness the fact that the guess is wrong — effectively making it right. Note that if we do not disable them then one of these processes will be in the wrong state (since in the AR system one of them must receive \mathbf{m} and move, but in the simulating system none moved) and will be able to later allow moves in the simulating system that are not possible in the simulated one. \square

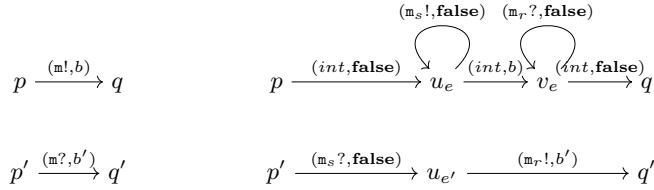


Fig. 6: Simulation of BCML (left) by PR (right)

Proposition 2. $\text{BCML} \leq_x \text{PR}$, for $x \in \{\text{FE}, \text{IE}\}$.

Proof. Given a local process P , we build a local process Q such that Q_{AR}^n simulates P_{PR}^n (see Figure 6). A global transition where i sends \mathbf{m} by taking a local transition of the form $e := (p, (\mathbf{m}!, b), q)$, and a set R of processes receive, is simulated by a multi-step transaction. The transaction needs multiple steps because in PR only two processes move in every step. The main difficulty, and the reason the transaction is complicated, is that we must be careful not to introduce new executions that are not possible in the BCML system.

The simulation of sending the lossy broadcast message \mathbf{m} is done in three stages: (a) process i internally and invisibly moves from state p to the new intermediate state u_e ; state u_e has an invisible self-loop that sends message \mathbf{m}_s (indicating it is trying to simulate sending \mathbf{m}); the self-loop enables the message to be sent to an arbitrary number of processes; (b) process i internally moves from state u_e to the new intermediate state v_e with visibility b ; state v_e has an invisible self-loop that receives message \mathbf{m}_r ; the self-loop allows to acknowledge that an arbitrary number of processes have received the message; (c) process i internally and invisibly moves from state v_e to state q .

The simulation of receiving message \mathbf{m} , by taking a local transition of the form $e' := (p', (\mathbf{m}?, b'), q')$, is in two stages: (a) process j invisibly moves from state p' to the new intermediate state $u_{e'}$ receiving message \mathbf{m}_s . (b) process j moves from state $u_{e'}$ to state q' with visibility b' sending message \mathbf{m}_r .

Unfortunately, we can not guarantee that this transaction is atomic, i.e., that no other global transitions intertwine with the simulation of a single lossy broadcast. We can not even guarantee that if a process j has received a message \mathbf{m}_s from process k , then it is going to send \mathbf{m}_r to process k , and not to another

process. The solution is to consider the processes that performed the second stage with some process k as the ones which received the lossy broadcast message m from k . This works since, for each process j , the first stage of receiving a lossy broadcast message is invisible, and after that it can not do anything but participate in a second stage of receiving a message. \square

Proposition 3. $\text{PR} \leq_x \text{AR}$, for $x \in \{\text{FE}, \text{IE}\}$.

Proposition 4. $\text{DG} \leq_x \text{BCML}$, for $x \in \{\text{FE}, \text{IE}\}$.

Remark: There is a version of broadcast, lets call it ABC, where a process can broadcast a message even when no other process is able to receive it. All our results about BC hold also for ABC since $\text{BC} \equiv_x \text{ABC}$, for $x \in \{\text{FE}, \text{IE}\}$.

4 Model Checking Asynchronous-Rendezvous Systems

The theorem below states that model checking even the most basic liveness properties of AR systems is undecidable. The proof of the theorem is an adaptation of the one used in ([12]) to prove a similar result for BC. Unfortunately, there is a serious complication: ([12]) makes central use of the fact that BC systems can elect a controller, but AR systems are not powerful enough to do that.

Fortunately, we can make do with a temporary controller, which AR can elect: from the initial state a process can send the message “I am now the controller” and enter the initial state of the “controller” component of the process template. If later on another process sends this message then it becomes the new controller, and the current controller, who receives this message, enters a special state D , from which it can do nothing. Thus, there are never two controllers at the same time, and at most n controller switches in a system with n processes.

The ability of AR to elect a temporary controller allows us not only to prove the theorem below, it also allows us to later show (see Figure 4) that AR systems have an expressive power that is in between PR (which cannot elect even a temporary controller) and BC (which can elect a permanent controller). However, interestingly enough, this is only true for infinite traces. For finite traces, having a temporary controller, in contrast to a permanent one, provides no extra expressive power (see Figure 3).

Theorem 1. (i) *Model checking liveness properties of parameterised systems communicating via AR is undecidable.* (ii) *In particular, the following problem is undecidable: given local process P , decide if $\text{INFEXEC}(P_{\text{AR}}^{\infty})$ is empty or not.*

Proof. For the first item, it is enough to reduce the halting problem for input-free deterministic counter machines CM (which is undecidable [19]) to the existence of a run in an AR system P_{AR}^{∞} that visits a halting state infinitely often. It is convenient to assume that when (and if) the halting location is reached then the CM resets itself, i.e., it decrements all its counters until they become zero and then loops back to the initial state. The basic encoding for the simulation is from [12]. It uses one process called the *controller* to orchestrate the simulation

and store the line of the CM, and many *memory* processes. Each memory process stores one bit for each counter, and the value of a counter is the number of processes having a non-zero bit for it. Each process has a special dead state D , which once entered cannot be exited.

A process may, from the initial state, nondeterministically become either the temporary controller or a memory process. The transitions in a memory process are, for each counter $c \in C$: if the stored c -bit is 0, then it can send the message “inc(c)” and set the c -bit to 1; if the stored c -bit is 1, then it can send the message “notzero(c)” and leave c unchanged, or send the message “dec(c)” and set c to 0, or *receive* the message “iszero(c)” and go to state D . From every state of the controller there is a complementary send/receive transition as specified by the CM line that this state represents. Thus, for example, an “increment c ” is simulated by the controller receiving an “inc(c)” and moving to the next line of the CM (or to the state D , if the current command to simulate is not “increment c ”), and an “if $c = 0$ goto l_1 else goto l_2 ” command is simulated by the controller either receiving “notzero(c)” and moving to state l_2 ; or moving to state l_1 and sending an “iszero(c)” which, if counter c is zero, is not received, and otherwise is received by a memory process with a 1 c -bit which then enters state D .

It is not hard to show that P_{AR}^n can faithfully simulate the CM as long as the counters stay below $n - 1$. Thus, if the CM reaches the halting location h then there is an infinite run of P_{AR}^n , for a large enough n , in which the process playing the controller is in h infinitely often. For the reverse direction, the key point is that whenever the simulation makes an error (such as replacing the temporary controller in mid simulation, or having the controller guess a counter is zero when it is not, or when a memory process simulates a command that is not what the controller wants to simulate) one process dies (i.e., enters state D). Thus, since there are only finitely many processes participating in any execution of P_{AR}^∞ , in every infinite run of P_{AR}^∞ , from some point on, no more processes die, and thus from that point on the simulation is correct. It follows that if there is a run of P_{AR}^∞ in which a process is in state h infinitely often then the run of the CM reaches the halting location. This completes the sketch of the proof of the first item. The second item uses a standard trick (see e.g. [12]) of adding an extra counter that increases in every step, and gets reset only when the halting state is reached. Thus, the system will run out of its finite number of memory processes and hang unless the CM reaches h . \square

5 Absolute Expressive Power

Finite Executions. First note that, since every prefix of a finite run is a run, for every CP and P we have that $\text{FINEXEC}(P_{\text{CP}}^\infty)$ is prefix-closed.¹²

Proposition 5. *For every CP and prefix-closed regular language L there exists P such that $\text{FINEXEC}(P_{\text{CP}}^\infty) = L$.*

¹² Although distributed systems are routinely studied this way, one may also introduce final states to the local process and restrict to runs that end in final states.[16]

Proof. Transform an automaton for L into P by pushing letters into states (i.e., by changing L so that it remembers the last read input-letter in its state, labeling each state by this letter, and adjusting the initial states), and making all local transitions of P internal (i.e., not synchronisation transitions) and visible. \square

Proposition 6 (cf. [16]). (i) There is a P s.t. $\text{FINEXEC}(P_{\text{BC}}^\infty) = \text{pre}(\{a^n b^n \mid n \geq 1\})$. Moreover, BC can generate non-context free languages; (ii) None of our communication primitives can generate all prefix-closed context-free languages.

Theorem 2 ([17]). For every P , the language $\text{FINEXEC}(P_{\text{PR}}^\infty)$ is regular.

Theorem 3. For every P , the language $\text{FINEXEC}(P_{\text{AR}}^\infty)$ is regular.

Proof. Let $P = \langle \Omega, A, S, S_0, \delta, \lambda \rangle$ be some local process. We will construct a finite automaton (NFW) \mathcal{A} that accepts exactly the traces in $\text{FINEXEC}(P_{\text{AR}}^\infty)$. We call a local state $s \in S$ *unbounded* if for every $k \in \mathbb{N}$ there is an $n \in \mathbb{N}$, and a reachable global state f in P_{AR}^n , such that $|f^{-1}(s)| \geq k$. We denote by $U \subseteq S$ the set of *unbounded states* of P and by $B = S \setminus U$ the set of *bounded states*. Observe that $S_0 \cap B = \emptyset$, and that there is a $K \in \mathbb{N}$ such that $|f^{-1}(s)| \leq K$ for every $s \in B$, and every global state f in P_{AR}^∞ .

We now define an automaton \mathcal{A} . States of \mathcal{A} are pairs $\langle s, f \rangle$, where $s \in S$ is the state of the process whose execution we are observing, and $f \in S \rightarrow \{0, 1, \dots, K\} \cup \{\infty\}$, is such that $f(u) = \infty$ for every $u \in U$. Intuitively, for each state in B , f keeps track of the number the other processes in that state. A state $\langle s, f \rangle$ of \mathcal{A} is initial iff: $s \in S_0$ and $f(u) = 0$ for all $u \in B$. \mathcal{A} has a transition from $\langle s, f \rangle$ to $\langle s', f' \rangle$ if there is a local transition $\tau \in \delta$ where the counter values of f change to f' according to τ (and any possible matching transition if τ is a synchronising transition), and if s is involved in τ then s changes to s' . Such a transition is labeled by $\lambda(s)$ if s was involved in the transition and $\text{vis}(\tau) = \text{true}$, and otherwise by ϵ . For example, if $\tau = (p, (\text{m}?, \text{true}), p')$ then, together with the any transition of the form $(q, (\text{m}!, b), q')$ in δ , it induces the following transitions in \mathcal{A} : (i) a transition $\langle p, f \rangle \xrightarrow{\lambda(p)} \langle p', f' \rangle$ for every f, f' such that $f(q) \neq 0$, and f' is obtained from f by decrementing the value assigned to q and incrementing the value assigned to q' ; (ii) a transition $\langle s, f \rangle \xrightarrow{\epsilon} \langle s, f' \rangle$ for every s and every f, f' such that $f(p) \neq 0, f(q) \neq 0$, and f' is obtained from f by decrementing the values assigned to p, q and incrementing the values assigned to p', q' (as usual, $\infty - 1 = \infty = \infty + 1$).

Clearly, \mathcal{A} is a finite automaton. We show that \mathcal{A} (with all states accepting) accepts exactly the traces in $\text{FINEXEC}(P_{\text{AR}}^\infty)$. For every $n \in \mathbb{N}$, every execution obtained from some path of $\text{FINEXEC}(P_{\text{AR}}^n)$ is accepted by a run of \mathcal{A} that “simulates” this execution by correctly updating the components s, f of its states.

It remains to prove that every word accepted by \mathcal{A} is in $\text{FINEXEC}(P_{\text{AR}}^\infty)$. We claim (*): for every k there exists $n_k \in \mathbb{N}$ and a path π_k of $\text{FINEXEC}(P_{\text{AR}}^{n_k})$ reaching a global state such that there are at least k processes in every local state $s \in U$. To see that (*) yields $L(\mathcal{A}) \subseteq \text{FINEXEC}(P_{\text{AR}}^\infty)$, let π be some run of \mathcal{A} , and take $k \geq 2|\pi|$. Observe that in such a run at most k processes are involved. We build a corresponding run in $P_{\text{AR}}^{n_k}$. First, (†): using (*) we take a

path that results in at least k processes in every local state $s \in U$. Recall that $S_0 \cap B = \emptyset$ and thus, in particular, there is at least one process in each of the initial states. Then, (‡): the process we want to observe starts from the relevant initial state and we imitate the run π of \mathcal{A} step by step. This is indeed possible since whenever a step of ‡ requires a process with a state in U then such a process is available, and the same for processes in B . The former is guaranteed by ‡, and the latter since (by induction on the step number) the number of processes with states in B is at least as specified by the function f of the mimicked point in π .

We now prove (*). Let u_1, \dots, u_m be the states in U . Inducting on $0 \leq i \leq m$, we construct paths π^i in systems $\text{FINEXEC}(P_{\text{AR}}^{n_i})$ such that load at least k processes in states u_1, \dots, u_i . We start with the empty run π^0 in $\text{FINEXEC}(P_{\text{AR}}^1)$. Clearly, π^0 satisfies the inductive claim. Given π^i , we construct π^{i+1} as follows: Let l_i be the length of π^i . By the definition of U , there is a path π in some system $\text{FINEXEC}(P_{\text{AR}}^n)$ that ends with at least $l_i + k$ processes in state u_{i+1} and at least one process in each of the initial states (thus, executing π in a larger system does not force any of the additional processes out of the initial states). We set $n_{i+1} = n + n_i$ and define π^{i+1} to be the concatenation of π and π^i in the system $\text{FINEXEC}(P_{\text{AR}}^{n_{i+1}})$. Clearly, π^{i+1} loads at least k processes in states u_1, \dots, u_{i+1} (since π^i can remove at most l_i states from u_{i+1}). \square

It is open if there is a constructive proof of Theorem 3.

Infinite Executions.

Theorem 4. *For every co-Büchi language L , and for $\text{CP} \in \{\text{PR}, \text{BCML}, \text{AR}, \text{BC}\}$, there is a local process P s.t. $\text{INFEXEC}(P_{\text{CP}}^\infty) = L$.*

Proof. Given an NCW \mathcal{A} recognizing L we build a local process P , in which all transitions are visible, such that $\text{INFEXEC}(P_{\text{CP}}^\infty) = L$. The local process P has exactly the same structure, when viewed as a graph, as \mathcal{A} , with an added special sink state. In order to take a transition to a co-Büchi state (i.e., a state that an accepting run of \mathcal{A} can only visit finitely many times) the process has to receive a message. A process that sends a message enters the sink state, and can not send again. Thus, in a system with n processes a process can visit a co-Büchi state up to $n - 1$ times. Transitions to other states are internal transitions of the process, and can always be taken. \square

We now show that not all ω -regular languages can be generated by our parameterised systems. In fact, the proof is general enough to apply to any reasonable notion of communication primitive (not only those defined in this paper), unless some additional fairness conditions are imposed. The proof employs a standard pumping argument to derive a contradiction by showing that if $ab^1ab^2a\dots$ is in $\text{EXEC}(P_{\text{CP}}^n)$ then so is $ab^1ab^2\dots ab^c ab^\omega$, where c is the number of states in P_{CP}^n .

Proposition 7. *For every local process P , primitive CP , and $n \in \mathbb{N} \cup \{\infty\}$, the set $\text{INFEXEC}(P_{\text{CP}}^n)$ is not equal to the ω -regular language $L \subseteq \{a, b\}^\omega$ consisting of all infinite sequences that contain infinitely many occurrences of a .*

The following is not hard to see:

Proposition 8. *There is a BC-system that can generate the non co-Büchi language $\{a^l b^m c^\omega \mid l \geq m \geq 1\}$.*

We use a variation of the proof of Theorem 1 to show that AR-systems can generate languages that are not ω -regular:

Proposition 9. *There is an AR-system that can generate a language $L \subset \{a, b\}^\omega$ that has the property that*

1. *every string $\alpha \in L$ has a suffix $(a^n b^n)^\omega$ for some integer $n \in \mathbb{N}$, and*
2. *every string $(a^n b^n)^\omega$ is the suffix of some string in L .*

In particular, the language L is not co-Büchi.

Proof. Standard fooling arguments show that any L with the properties described is not Büchi (and thus not co-Büchi). We now describe an AR-system that can generate a language L with the properties stated in the lemma. The idea follows that in the proof of Theorem 1: a controller starts in mode a ; in mode a it repeatedly increments a counter c ; at some point it checks if all memory processes are 1 by issuing an “allone(c)” message (which can be implemented symmetric to the “iszero(c)” message), and moves to mode b ; in mode b it repeatedly decrements the counter c ; at some point it checks if all memory processes are 0 by issuing a “iszero(c)” message, and moves back to mode a to repeat the computation. Build the local process P based on M and note that a process that becomes a controller forever in P_{AR}^l does not err from some point on, and thus traces a path whose suffix is $(a^n b^n)^\omega$ with $n \leq l$. Note that an abdicating controller does not trace an infinite path (since the dead state is a dead-end). \square

The following is proved in almost the same way as Theorem 3:

Theorem 5. *For every P , the language $\text{pre}(\text{INFEXEC}(P_{\text{AR}}^\infty))$ is regular.*

Model checking safety and liveness properties (given as automata) of parameterised systems communicating via PR is decidable in PTIME [17]. Actually:

Theorem 6 (implicit in [17]). *For every local process P , one can compute, in PTIME, a non-deterministic co-Büchi automaton for the set $\text{INFEXEC}(P_{\text{PR}}^\infty)$.*

Theorem 7. *Every $\text{INFEXEC}(P_{\text{DG}}^\infty)$ is the limit of a regular language.*

Proof. By [10] there exists $N \in \mathbb{N}$ such that $\text{INFEXEC}(P_{\text{DG}}^\infty) = \text{INFEXEC}(P_{\text{DG}}^N)$ (the idea is to pick N large enough such that every reachable state can be reached and adding one extra process; this choice of N ensures that every reachable self-loop $(s, (\text{!}, \text{false}), s)$ can always be fired; the extra process can therefore move unrestrained). The language $L := \text{FINEXEC}(P_{\text{DG}}^N)$ is regular (because it is the projection of the finite-state machine P_{DG}^N). It is sufficient to prove that $\text{INFEXEC}(P_{\text{DG}}^N) = \lim L$. Clearly $\text{INFEXEC}(P_{\text{DG}}^N) \subseteq \lim L$. To see the converse let $\alpha \in \lim L$. So there exists $k \in [N]$ and an infinite set $I \subseteq \mathbb{N}$ such that for every $i \in I$ there exists a run ρ_i of P_{DG}^N such that the prefix of α of length i is equal to $\text{vislet}_k(\rho_i)$. The set $\text{pre}\{\rho_i : i \in I\}$ is an infinite tree (under the prefix-ordering) that is finitely-branching (this is where we use the fact that the ρ_i s are in P_{DG}^N and not P_{DG}^∞), and thus by König’s Lemma, it has an infinite branch ρ . Clearly ρ is an infinite run of P_{DG}^N and $\text{vislet}_k(\rho) = \alpha$. Thus $\alpha \in \text{INFEXEC}(P_{\text{DG}}^N)$. \square

6 Related Work and Conclusion

Related Work. The absolute and relative expressive power of Petri nets and their extensions were studied for finite and infinite executions, e.g., [14,16,1]. They show a strict hierarchy of relative expressive power: Petri nets (PN) are less expressive than Petri nets with non-blocking arcs (PN+NBA), which are less expressive than Petri nets with transfer arcs (PN+T). Translating these results into the language of parameterised systems, one finds that these extensions roughly correspond to a very powerful model of parameterised systems with a controller and in which processes can be created and destroyed at any time. By this translation, PNs correspond to communication by PR, PN+NBA to communication by AR, and PN+T to communication by BC. In contrast, we focus on the setting with no controller and with no process creation or destruction. Thus, neither their simulation nor separation results are directly applicable to our more restricted setting.

The paper [9] organises communication primitives by whether or not model checking (MC) is decidable. Although they do have a notion of simulation, that notion is based on reducing the MC problem of systems using one primitive to systems using another primitive. In particular, their reduction transforms, while ours preserves, the set of behaviours. For instance, despite their result that MC safety properties of DG- and PR-systems are inter-reducible, we prove that there is a set of traces of a PR system that can't be generated by any DG system.

It was previously known that MC safety properties for systems using each of the primitives in this paper is decidable, liveness for BC is undecidable, and liveness for PR and DG is decidable [12,17,10,6,9]. We complete the picture, and prove, in particular, that for AR systems liveness is undecidable. The result in [9] on the undecidability of liveness for AR systems makes the additional assumption that there exists a unique “leader” process. The presence of a leader usually dramatically increases the expressive power, cf. [11,17], and makes it easier to establish undecidability than in our fully symmetric case. A number of papers focus on supplying the exact complexity of MC various parameterised systems, e.g., [11,2,17,6,7,8,21].

Conclusion. Comparing the expressive power of various models of computation is a central theme in theoretical computer science. In our case, such comparisons can be used to transfer results from one model to another. For instance, we prove that AR can be effectively simulated by BC, and thus the fact that safety is decidable for BC (cf. [12]) implies that safety is decidable for AR ([9]). We also deduced the new result, using [17] and the fact that BCML can be efficiently simulated by PR, that liveness for BCML is decidable in PTIME.

The results about absolute expressive power are useful not only to show, e.g., that PR can not simulate AR, but also to point to the inherent limitations of each communication primitive. Such results can be used in synthesis to show that certain specifications are not realisable. As a concrete example, a minor variation of our proof that no system can generate the language “infinitely many a 's” (Proposition 7) yields that there is no parameterised system (and thus no

point in trying to synthesise one without adding external fairness conditions) that satisfies the conjunction of the properties “every run has infinitely many grants” and “some run has arbitrarily large gaps between successive grants”.

References

1. Abdulla, P.A., Delzanno, G., Begin, L.V.: A classification of the expressive power of well-structured transition systems. *Inf. Comput.* 209(3), 248–279 (2011)
2. Aminof, B., Kotek, T., Rubin, S., Spegni, F., Veith, H.: Parameterized model checking of rendezvous systems. In: CONCUR, pp. 109–124. Springer (2014)
3. Aminof, B., Rubin, S., Zuleger, F., Spegni, F.: Liveness of parameterized timed networks. In: ICALP. pp. 375–387 (2015)
4. Aspnes, J., Ruppert, E.: An introduction to population protocols. In: Middleware for Network Eccentric and Mobile Applications, pp. 97–120. Springer-Verlag (2009)
5. Delzanno, G., Raskin, J.F., Begin, L.V.: Towards the automated verification of multithreaded java programs. In: TACAS. pp. 173–187 (2002)
6. Delzanno, G., Sangnier, A., Traverso, R., Zavattaro, G.: The cost of parameterized reachability in mobile ad hoc networks. *CoRR* abs/1202.5850 (2012)
7. Delzanno, G., Sangnier, A., Zavattaro, G.: Parameterized verification of ad hoc networks. In: CONCUR. LNCS, vol. 6269, pp. 313–327 (2010)
8. Delzanno, G., Sangnier, A., Zavattaro, G.: Verification of ad hoc networks with node and communication failures. In: FTDS. pp. 235–250 (2012)
9. Emerson, E., Kahlon, V.: Model checking guarded protocols. In: LICS. pp. 361–370. IEEE (2003)
10. Emerson, E., Kahlon, V.: Reducing model checking of the many to the few. In: CADE. pp. 236–254 (2000)
11. Esparza, J.: Keeping a crowd safe: On the complexity of parameterized verification. In: STACS (2014)
12. Esparza, J., Finkel, A., Mayr, R.: On the verification of broadcast protocols. Symp. on Logic in Computer Science p. 352 (1999)
13. Esparza, J., Ganty, P., Majumdar, R.: Parameterized verification of asynchronous shared-memory systems. In: CAV. pp. 124–140 (2013)
14. Finkel, A., Geeraerts, G., Raskin, J., Begin, L.V.: On the *omega*-language expressive power of extended petri nets. *Theor. Comput. Sci.* 356(3), 374–386 (2006)
15. Fisher, J., Henzinger, T.A.: Executable cell biology. *Nature biotechnology* 25(11), 1239–1249 (2007)
16. Geeraerts, G., Raskin, J., Begin, L.V.: Well-structured languages. *Acta Inf.* 44(3-4), 249–288 (2007)
17. German, S.M., Sistla, A.P.: Reasoning about systems with many processes. *J. ACM* 39(3), 675–735 (1992)
18. Lynch, N.: Distributed Algorithms. Morgan Kaufman Publishers, Inc., San Francisco, USA (1996)
19. Minsky, M.L.: Computation: finite and infinite machines. Prentice-Hall, Inc., Upper Saddle River, NJ, USA (1967)
20. Prasad, K.V.S.: A calculus of broadcasting systems. *Sci. Comput. Program.* 25(2-3), 285–327 (1995)
21. Schmitz, S., Schnoebelen, Ph.: The power of well-structured systems. In: CONCUR. pp. 5–24 (2013)
22. Vardi, M.Y.: An automata-theoretic approach to linear temporal logic. In: Banff Higher Order Workshop. pp. 238–266 (1995)

Decidability of Parameterized Verification

Synthesis Lectures on Distributed Computing Theory

Editor

Jennifer Welch, *Texas A&M University*

Nancy Lynch, *Massachusetts Institute of Technology*

Synthesis Lectures on Distributed Computing Theory is edited by Jennifer Welch of Texas A&M University and Nancy Lynch of the Massachusetts Institute of Technology. The series publishes 50- to 150-page publications on topics pertaining to distributed computing theory. The scope largely follows the purview of premier information and computer science conferences, such as ACM PODC, DISC, SPAA, OPODIS, CONCUR, DialM-POMC, ICDCS, SODA, Sirocco, SSS, and related conferences. Potential topics include, but are not limited to: distributed algorithms and lower bounds, algorithm design methods, formal modeling and verification of distributed algorithms, and concurrent data structures.

[Decidability of Parameterized Verification](#)

Roderick Bloem, Swen Jacobs, Ayrat Khalimov, Igor Konnov, Sasha Rubin, Helmut Veith, and Josef Widder

2015

[Impossibility Results for Distributed Computing](#)

Hagit Attiya and Faith Ellen

2014

[Distributed Graph Coloring: Fundamentals and Recent Developments](#)

Leonid Barenboim and Michael Elkin

2013

[Distributed Computing by Oblivious Mobile Robots](#)

Paola Flocchini, Giuseppe Prencipe, and Nicola Santoro

2012

[Quorum Systems: With Applications to Storage and Consensus](#)

Marko Vukolic

2012

[Link Reversal Algorithms](#)

Jennifer L. Welch and Jennifer E. Walter

2011

[**Cooperative Task-Oriented Computing: Algorithms and Complexity**](#)

Chryssis Georgiou and Alexander A. Shvartsman

2011

[**New Models for Population Protocols**](#)

Othon Michail, Ioannis Chatzigiannakis, and Paul G. Spirakis

2011

[**The Theory of Timed I/O Automata, Second Edition**](#)

Dilsun K. Kaynar, Nancy Lynch, Roberto Segala, and Frits Vaandrager

2010

[**Principles of Transactional Memory**](#)

Rachid Guerraoui and Michal Kapalka

2010

[**Fault-tolerant Agreement in Synchronous Message-passing Systems**](#)

Michel Raynal

2010

[**Communication and Agreement Abstractions for Fault-Tolerant Asynchronous Distributed Systems**](#)

Michel Raynal

2010

[**The Mobile Agent Rendezvous Problem in the Ring**](#)

Evangelos Kranakis, Danny Krizanc, and Euripides Markou

2010

Copyright © 2015 by Morgan & Claypool

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means—electronic, mechanical, photocopy, recording, or any other except for brief quotations in printed reviews, without the prior permission of the publisher.

Decidability of Parameterized Verification

Roderick Bloem, Swen Jacobs, Ayrat Khalimov, Igor Konnov, Sasha Rubin, Helmut Veith, and Josef Widder

www.morganclaypool.com

ISBN: 9781627057431 paperback

ISBN: 9781627057448 ebook

DOI 10.2200/S00658ED1V01Y201508DCT013

A Publication in the Morgan & Claypool Publishers series

SYNTHESIS LECTURES ON DISTRIBUTED COMPUTING THEORY

Lecture #13

Series Editors: Jennifer Welch, *Texas A&M University*

Nancy Lynch, *Massachusetts Institute of Technology*

Series ISSN

Print 2155-1626 Electronic 2155-1634

Decidability of Parameterized Verification

Roderick Bloem and Ayrat Khalimov
Graz University of Technology, Austria

Swen Jacobs
Graz University of Technology, Austria
Saarland University, Saarbrücken, Germany

Igor Konnov, Helmut Veith, and Josef Widder
Vienna University of Technology, Austria

Sasha Rubin
University of Naples “Federico II”, Italy

SYNTHESIS LECTURES ON DISTRIBUTED COMPUTING THEORY #13



MORGAN & CLAYPOOL PUBLISHERS

ABSTRACT

While the classic model checking problem is to decide whether a finite system satisfies a specification, the goal of parameterized model checking is to decide, given finite systems $\mathbf{M}(n)$ parameterized by $n \in \mathbb{N}$, whether, for all $n \in \mathbb{N}$, the system $\mathbf{M}(n)$ satisfies a specification. In this book we consider the important case of $\mathbf{M}(n)$ being a concurrent system, where the number of replicated processes depends on the parameter n but each process is independent of n . Examples are cache coherence protocols, networks of finite-state agents, and systems that solve mutual exclusion or scheduling problems. Further examples are abstractions of systems, where the processes of the original systems actually depend on the parameter.

The literature in this area has studied a wealth of computational models based on a variety of synchronization and communication primitives, including token passing, broadcast, and guarded transitions. Often, different terminology is used in the literature, and results are based on implicit assumptions. In this book, we introduce a computational model that unites the central synchronization and communication primitives of many models, and unveils hidden assumptions from the literature. We survey existing decidability and undecidability results, and give a systematic view of the basic problems in this exciting research area.

KEYWORDS

parametrized model checking, concurrent systems, distributed systems, formal verification, model checking, decidability, cutoffs

Contents

Acknowledgments	xi
1 Introduction	1
1.1 Motivation	2
1.2 Who Should Read This Book?	4
1.3 Organization of the Book	4
2 System Model and Specification Languages	5
2.1 Preliminary Terminology and Definitions	5
2.2 System Model	6
2.2.1 Some Standard Synchronization Primitives	9
2.2.2 Runs and Deadlocks	12
2.3 Parameterized Family of Uniform Concurrent Systems	13
2.4 Parameterized Specifications	13
2.4.1 Indexed Temporal Logics	14
2.4.2 Action-based Specifications	18
2.4.3 Specifications in the Literature	18
2.5 Model Checking Problems for Concurrent Systems	19
2.5.1 Computability Assumptions	20
3 Standard Proof Machinery	23
3.1 Techniques to Prove Undecidability of PMCP	23
3.2 How to Prove Decidability of the PMCP	24
3.2.1 Well-structured Transition Systems	25
3.2.2 Vector Addition Systems with States (and Petri Nets)	27
3.2.3 Decompositions and Cutoffs	28
4 Token-passing Systems	31
4.1 System Model	32
4.1.1 Direction-aware Parameterized Systems	32
4.1.2 Token-passing Systems	34
4.2 Results for Direction-unaware Token-passing Systems	37

4.2.1	Decidability for Simple Token-passing in Uni-directional Rings	38
4.2.2	Decidability for Simple Token-passing in Graphs	40
4.2.3	Undecidability Results for Multi-valued Tokens	42
4.3	Results for Direction-aware Token-passing Systems	44
4.3.1	Cutoffs for Change-bounded Tokens in Bi-directional Rings	44
4.3.2	Undecidability for Direction-aware TPSs	45
4.4	Discussion	46
4.4.1	Variations of the Model	48
5	Rendezvous and Broadcast	51
5.1	System Model	51
5.2	Decidability Results	55
5.2.1	Counter Representation	55
5.2.2	Decidability for All Three Primitives	58
5.2.3	Decidability for Pairwise Rendezvous	58
5.3	Undecidability Results	59
5.3.1	Undecidability for Broadcast	59
5.3.2	Undecidability for Asynchronous Rendezvous	62
5.4	Discussion	62
5.4.1	Variations of the Model	64
6	Guarded Protocols	65
6.1	Motivating Example	65
6.2	System Model	69
6.2.1	Classes of Guarded Protocols	73
6.2.2	Specifications	74
6.3	Undecidability: Boolean and Conjunctive Guards	76
6.4	Decidability: Init-Conjunctive and Disjunctive Guards	81
6.4.1	Preliminaries	82
6.4.2	Proof Schemas	84
6.4.3	Init-conjunctive Guards	86
6.4.4	Disjunctive Guards	89
6.5	Disjunctive Guards vs. Rendezvous	96
6.6	Variations on the Model: Guards and Synchronization Primitives	99
6.7	Discussion	100

7	Ad Hoc Networks	103
7.1	Running Example	103
7.2	System Model	104
7.3	PMC Problems for Ad Hoc Networks	105
7.4	Parameterized Connectivity Graphs $\mathbf{BP}_k, \mathbf{BPC}_k, \mathbf{BD}_k, \mathbf{C}, \mathbf{All}$	107
7.5	Results for (Non-lossy) AHNs	108
7.5.1	Undecidability Results for (Non-lossy) AHNs	108
7.5.2	Decidability Results for (Non-lossy) AHNs	116
7.6	Decidability Results for Lossy Ad Hoc Networks	123
7.7	Discussion	127
7.7.1	Variations of the Model	128
8	Related Work	129
8.1	Abstraction Techniques	129
8.2	Regular Model Checking	131
8.3	Symbolic Techniques	132
8.4	Dynamic Cutoff Detection	134
8.5	Network Invariants	135
8.6	Invisible Invariants	137
8.7	Other Aspiring Approaches	137
9	Parameterized Model Checking Tools	139
10	Conclusions	143
	Bibliography	145
	Authors' Biographies	157

Acknowledgments

We are grateful to Paul Attie, Giorgio Delzanno, Sayan Mitra, and Kedar Namjoshi for carefully reading an earlier draft of this manuscript, and providing detailed and constructive comments.

Supported by the Austrian National Research Network RiSE (S11403, S11405, S11406) and project PRAVDA (P27722) of the Austrian Science Fund (FWF), by the Vienna Science and Technology Fund (WWTF) through grant PROSEED, by the German Research Foundation (DFG) through SFB/TR 14 AVACS and project ASDPS (JA 2357/2-1), and by the Istituto Nazionale di Alta Matematica through INdAM-COFUND-2012, FP7-PEOPLE-2012-COFUND (Proj. ID 600198).

Roderick Bloem, Swen Jacobs, Ayrat Khalimov, Igor Konnov, Sasha Rubin, Helmut Veith, and Josef Widder
August 2015

CHAPTER 1

Introduction

Reasoning about the correctness of concurrent and distributed systems is an inherently difficult task. A main challenge comes from the fact that many processes run concurrently and interact with each other, which results in non-determinism and large execution and state spaces. Thus, it is easy for a system designer to miss a bug related to concurrency, e.g., an unforeseen race condition, deadlock, livelock, etc. Finding these kinds of bugs motivated research in model checking for many years [Baier and Katoen, 2008, Clarke et al., 1999, Emerson and Clarke, 1980, Grumberg and Veith, 2008, Queille and Sifakis, 1982]. The central problem for model checking (and of many other techniques) is state explosion. Research in model checking made several breakthroughs and developed methods to deal with this problem. Over the years, these methods have been implemented in industrial tools, which are used to verify hardware designs (e.g., microprocessors and cache coherence protocols), predominantly sequential software (e.g., device drivers), and network protocols [Grumberg and Veith, 2008]. The main line of research in model checking of concurrent systems considers systems with a fixed and *a priori* known number of processes.

Some designs, however, need to be proven to work independently of the system size (e.g., hardware protocols, network protocols, distributed algorithms). The engineering intuition says that if a system has been carefully debugged on a small number of processes, then it should also work with a large number of processes. This intuition is sometimes wrong. For instance, Clarke et al. [1992] applied model checking on the Futurebus+ cache coherence protocol for small sizes. After discovering several bugs, Clarke et al. proposed an improved version of the protocol, and verified that the new version of the protocol is correct on various instances of small size. Later, Kesten et al. [1997] performed model checking on the new protocol for larger system sizes and discovered new bugs.

This book focuses on the verification of systems where the number of processes is not known *a priori*. Because the number of processes is not known, the challenge is to verify a system for all system sizes. This motivates the study of *parameterized model checking*. In this book we survey results regarding the decidability of this problem for concurrent systems, and we thus make explicit the limits and strengths of automated verification methods. For instance, we will see cases where one can prove that it is sufficient to verify only system instances of small size, that is, special cases where the intuition mentioned above is actually justified. However, we will also see that parameterized model checking is undecidable for many interesting classes.

These decidability results, and the border between decidability and undecidability of the parameterized model checking problem, are the topics of this book. In the decidable cases we,

2 1. INTRODUCTION

in general, do not cover complexity results. Many existing formalisms (e.g., Petri-nets, process algebra) are related to concurrent systems, and complexity results regarding these formalisms may carry over to the parameterized model checking problem. As there is a wealth of complexity results related to such formalisms we cannot cover them in this book. Also, we do not cover infinite-state systems such as automata with unbounded channels or external storage, or concurrent systems in all generality. We focus on concurrent systems that are composed of a parameterized number of identical finite-state processes, where the local state space is independent of the parameter. Consequently, each system instance we consider has a finite state space.

The rationale of this book is to give an overview of the positive and negative results, and to ease the understanding by expressing system models and proofs in a unified model. Thereby, we hope to shed some light on the border between decidability and undecidability of the parameterized model checking problem.

1.1 MOTIVATION

If the number of components in a concurrent system is known a priori and every component has finitely many states, then the verification problem corresponds to finite-state *model checking* [Baier and Katoen, 2008, Clarke et al., 1999]: given the specification as a temporal logic formula ϕ , and given the system with n components as a finite-state transition system M , check whether M satisfies ϕ . In practice, reasoning about properties of concurrent systems is complicated because the interleaving of steps leads to combinatorial explosion.

Many protocols like cache coherency or bus protocols are defined as *parameterized* systems. Intuitively, a parameterized system is a sequence $(\mathbf{M}(n))_{n \in \mathbb{N}}$ of systems, where $\mathbf{M}(n)$ consists of n copies of similar or identical processes. The *parameterized model-checking problem* (PMCP) is then to decide whether $\mathbf{M}(n)$ satisfies the specification for every n . We survey the work in this area, focusing on decidability and undecidability results, and putting less emphasis on pragmatic solutions for real-life systems.

A tempting approach to the PMCP is to check the system for small values of n and *assume* that if there is a bug in a system with a large number of components, then it already appears in a small system. This assumption is widely used; for instance, Lamport [2006, p. 13] writes “Almost every error manifests itself on a very small instance—one that may or may not be too large to model check.” For specific classes of systems, this approach can be formalized as a *cutoff* or *decomposition* statement [Clarke et al., 2004, Emerson and Namjoshi, 1995] that reduces the PMCP to a finite collection of classic model checking problems.

Unfortunately, not all bugs of large systems can be found in small ones, and the PMCP is undecidable in general. Indeed, Apt and Kozen [1986] formalized the PMCP and were the first to address the question of its decidability. However, they considered systems where the implementation of a single process S depends on the parameter n , and treated concurrency only superficially. To prove undecidability of the PMCP, Apt and Kozen considered the following program: Given

a deterministic Turing Machine T , let $S(n)$ be the finite-state system represented by the following code:

```
flag := false
for i := 1 to n do
    simulate one step of T
if T has not halted then flag := true
```

Hence, the system $S(n)$ simulates the first n steps of the Turing machine T . Let ϕ be the formula stating that eventually the flag is set to true—in linear temporal logic this is written as $\mathbb{F}flag$. Then T does not halt *if and only if* $\forall n \in \mathbb{N}. S(n) \models \phi$. As the non-halting problem is undecidable, one immediately finds the PMCP to be undecidable. Furthermore, as $S(n)$ is a degenerate case of a concurrent system (consisting of only one process), one may conclude that the PMCP for concurrent systems is undecidable in general.

However, the undecidability result by [Apt and Kozen \[1986\]](#) does not directly apply to a particularly interesting class of concurrent systems, namely those that are composed of n copies of a finite-state process P , where P is independent of n . We call these *uniform concurrent systems*. Examples of such systems solve classic problems such as mutual exclusion [[Pnueli et al., 2002](#), [Wolper and Lovinfosse, 1989](#)], or scheduling [[Milner, 1989](#)]. For specific fault-tolerant distributed broadcasting algorithms [[Srikanth and Toueg, 1987](#)], [John et al. \[2013\]](#) obtained a uniform concurrent system after a data abstraction of the parameterized process code of the broadcasting algorithm. Hence, PMCP of specific uniform concurrent systems can also be used as part of a tool chain in the verification of concurrent systems where P actually depends on n .

Whether the PMCP for a uniform concurrent system is decidable depends on several factors, the most important being the underlying communication graph (e.g., rings, stars, cliques), and the means of synchronization (e.g., token passing with/without information-carrying tokens, handshake). For instance, [Suzuki \[1988\]](#) proved that [Apt and Kozen](#)'s idea of reducing the PMCP to the non-halting problem can be extended to token rings in which tokens carry information. The basic idea is to use each of the n replicated finite-state processes to store the content of one cell of the tape of a Turing machine, and implement the movement of the head by shifting information around the ring using tokens. As n cells can be stored in a system with n processes, such a system can be used to simulate n steps of a Turing machine, and undecidability follows as in [Apt and Kozen \[1986\]](#). [Suzuki](#)'s construction is prototypical of the undecidability results surveyed in this book.

However, the parameterized model checking results in the literature are not limited to token rings. Rather, there are many different computational models that differ in the underlying graph, and the means of communication. These computational models are scattered over the literature, use different terminology, with slightly different assumptions.

Hence, if one is faced with a PMCP in a given system, it is difficult to tell whether there is a published computational model that naturally captures the system's semantics. The main goal of this work is to provide a *one-stop source* for existing models for asynchronous *uniform concurrent*

4 1. INTRODUCTION

systems, along with known (un)decidability results. To this end, we provide a single definition that incorporates many of the foundational computational models that have appeared in the parameterized model checking literature on undecidability and decidability. The later chapters then specialize specific features of the general model to systems that can be found in the literature, and discuss the applicable (un)decidability results.

1.2 WHO SHOULD READ THIS BOOK?

This book was written for people who are interested in the principles of concurrent systems and parameterized model checking. This includes graduate students and senior undergraduate students in computer science who are interested in the basic principles of parameterized model checking for concurrent systems. For researchers this book surveys the principles that have been used to obtain decidability and undecidability results in the context of parameterized model checking. We are convinced that many of the techniques surveyed in this book can be used to derive new results in the field.

1.3 ORGANIZATION OF THE BOOK

In Chapter 2 we introduce definitions of a computational model and parameterized specifications that cover most of the existing decidability results in parameterized model checking. In Chapter 3 we explain foundational ideas from the literature for proving decidability or undecidability of different subclasses of the PMCP.

Subsequently, we consider four classes of systems where decidability results have been obtained.

1. The well-studied class of *token-passing systems*, where communication is restricted to passing a single token among all processes. This includes the important special case of *token rings* (Chapter 4).
2. Systems where communication is based on synchronized steps between two processes, called *pairwise rendezvous*, or one-to-many synchronization, called *broadcast* (Chapter 5).
3. *Guarded protocols*, where local transitions of one process can be restricted with respect to the global state of the system (Chapter 6). This includes combinations with other forms of synchronization, like broadcast and rendezvous.
4. *Ad-hoc networks*, where broadcast communication between processes may be lossy, or equivalently, communication networks can change during runtime (Chapter 7).

For all classes, we survey decidability and undecidability results from the literature. After briefly discussing related work that is not surveyed in this book in Chapter 8 and giving an overview of existing parameterized model checking tools in Chapter 9, we conclude in Chapter 10.

CHAPTER 2

System Model and Specification Languages

In this chapter we present definitions of parameterized systems, parameterized specifications and the parameterized model-checking problem. The rationale behind our definitions is to make explicit many of the commonalities of the different system models and results found in the literature.

A concurrent system \overline{P}^G (called a *system instance*) is composed of a vector of *process templates* $\overline{P} = (P^1, \dots, P^d)$, with copies of the templates arranged on a graph G (called the *connectivity graph*). We consider discrete time, and at each time step either one process acts alone, or some process v initiates an action and some set of processes that are connected to v in G simultaneously synchronize with v .

Synchronization primitives result from restricting the number of processes that can simultaneously synchronize with the initiating process. We capture a given synchronization primitive (e.g., pairwise rendezvous, broadcast) by a *synchronization constraint card* $\subseteq \mathbb{N}_0$. Roughly, the number of synchronizing processes should be a number in card. For instance, pairwise rendezvous is captured by defining $\text{card} = \{1\}$: every synchronous transition must be taken by the initiating process and exactly one other process.

We define \overline{P}^G as a labeled transition system, and thus need to specify its labeling function. If p is an atomic proposition of the process template and v is a vertex in G , then p_v is an atomic proposition of the composed system—an *indexed atomic proposition*—meaning that p holds in the current state of the process at vertex v .

Then, a *parameterized system* is a sequence of system instances formed from a fixed vector of process templates and a sequence of graphs \mathbf{G} . Typical sequences of graphs are rings of size n , cliques of size n , or stars of size n , where n is the parameter. The n th instance of a parameterized system is the system instance $\overline{P}^{\mathbf{G}(n)}$. *Parameterized specifications* make statements about parameterized systems by quantifying over process indices (i.e., vertices of $\mathbf{G}(n)$), e.g., “every process v of $\mathbf{G}(n)$ eventually satisfies p .” The *parameterized model checking problem* is to decide whether for all $n \in \mathbb{N}$ the instance $\overline{P}^{\mathbf{G}(n)}$ satisfies the specification.

2.1 PRELIMINARY TERMINOLOGY AND DEFINITIONS

A *labeled transition system (LTS)* over AP is a tuple $(Q, Q_0, \Sigma, \delta, \lambda)$, where AP is a set of *atomic propositions* or *atoms*, Q is the set of *states*, $Q_0 \subseteq Q$ are the *initial states*, Σ is the set of *transition*

6 2. SYSTEM MODEL AND SPECIFICATION LANGUAGES

labels (also-called *action labels*), $\delta \subseteq Q \times \Sigma \times Q$ is the *transition relation*, and $\lambda : Q \rightarrow 2^{\text{AP}}$ is the *state-labeling* and satisfies that $\lambda(q)$ is finite (for every $q \in Q$). A *finite LTS* is an LTS in which AP and Q are finite. Transitions $(q, a, q') \in \delta$ may be written $q \xrightarrow{a} q'$. A *transition system* is an LTS without the state-labeling λ , and sometimes without initial states.

A *path of an LTS* $(Q, Q_0, \Sigma, \delta, \lambda)$ is a finite sequence of the form $q_0 a_0 q_1 a_1 \dots q_n \in (Q \Sigma)^* Q$ or an infinite sequence of the form $q_0 a_0 q_1 a_1 \dots \in (Q \Sigma)^\omega$ such that $(q_i, a_i, q_{i+1}) \in \delta$ for all i . A *run of an LTS* is a path that starts in an initial state, i.e., $q_0 \in Q_0$. A *state-labeled path* of an LTS is the projection $q_0 q_1 \dots$ of a path onto states Q . An *action-labeled path* of an LTS is the projection $a_0 a_1 \dots$ of a path onto transition labels Σ .

When it is clear from the context, a path may omit either actions or states (e.g., LTL formulas are interpreted over infinite state-labeled paths $q_0 q_1 \dots$).

2.2 SYSTEM MODEL

Given a finite set of atomic propositions AP_{pr} (for individual processes), and process identifiers from the set \mathbb{N} , define the set of *indexed atomic propositions* $\text{AP}_{\text{sys}} := \text{AP}_{\text{pr}} \times \mathbb{N}$. The set of atomic propositions for a system instance with k processes is $\text{AP}_{\text{pr}} \times [k] \subseteq \text{AP}_{\text{sys}}$. For $(p, i) \in \text{AP}_{\text{sys}}$ we may also write p_i .

INGREDIENTS OF SYSTEM INSTANCE

Transition labels. Write Σ_{int} for a finite non-empty set of *internal transition labels* and Σ_{sync} for a finite non-empty set of *synchronous transition labels*. Throughout this book, when we use τ , we assume that is an action with $\tau \in \Sigma_{\text{int}}$. Define Σ_{pr} as the disjoint union $\Sigma_{\text{int}} \cup \{\text{out}_a : a \in \Sigma_{\text{sync}}\} \cup \{\text{in}_a : a \in \Sigma_{\text{sync}}\}$, where out_a and in_a are new symbols. An action out_a will be called an *initiate action* and an action in_a a *receive action*. Formally, in_a may be defined as $(a, 0)$ and out_a as $(a, 1)$.

System-arity d . Let d be a positive integer called the *system-arity* or just *arity*.

d -ary system template \bar{P} . A d -ary system template is a d -tuple $\bar{P} = (P^1, \dots, P^d)$ where each P^ℓ , called a *process template*, is a finite LTS $(Q^\ell, Q_0^\ell, \Sigma_{\text{pr}}, \delta^\ell, \lambda^\ell)$ over atomic propositions AP_{pr} . The elements of Q^ℓ are called *local states* — the Q 's are assumed to be pairwise disjoint — and the transitions in δ^ℓ are called *local transitions of P^ℓ* . Furthermore, we require the modest restriction that every δ^ℓ is total in the first coordinate: for every $q \in Q^\ell$ there exists $\sigma \in \Sigma_{\text{pr}}, q' \in Q^\ell$ such that $(q, \sigma, q') \in \delta^\ell$. In case $d = 1$ we write P instead of \bar{P} .

d -ary connectivity graph G . A d -ary connectivity graph is a d -colored directed graph $G = (V, E, \text{type})$ where $V = [k]$ for some $k \in \mathbb{N}$, $E \subseteq V^2$ and $\text{type} : V \rightarrow [d]$. Vertices are called *process indices*. For $i \leq d$ let n_i denote the cardinality of $\{v \in V : \text{type}(v) = i\}$. If $(v, w) \in E$ then we say that w is a *recipient of v* , and also write $w \in E(v)$. In case $d = 1$ we may write $G = (V, E)$ instead of $G = (V, E, \text{type})$. To distinguish vertices and edges of different graphs, we introduce the

notation $V(G)$ and $E(G)$ to refer to the set of vertices and the set of edges of graph G , respectively. A connectivity graph is sometimes also-called a *topology*.

Example 2.1 Simple Client/Server System. As an example, consider a simple client/server system consisting of a server and three clients. In our model, this is captured by a 2-ary connectivity graph with $V = \{1, 2, 3, 4\}$: vertex 1 is the server and thus $\text{type}(1) = 2$, and vertices 2, 3, and 4 are clients so that $\text{type}(i) = 1$ for $i \in \{2, 3, 4\}$. To model that only the clients can initiate synchronization, we set $E = \{(2, 1), (3, 1), (4, 1)\}$. The connectivity graph is depicted in Figure 2.1(a).

Now, suppose the implementations of server and client (as LTSs) are based on internal transition labels $\Sigma_{int} = \{\tau\}$ and synchronous transition labels $\Sigma_{sync} = \{\text{enter}, \text{leave}\}$. Very simple implementations S and C for the server and client, respectively, are depicted in Figure 2.1(b) and Figure 2.1(c) (omitting state labels).

In the composed system, client processes can take internal transitions (labeled with α) independently of the others, while for the other transitions they need to synchronize with the server. The details of synchronization are defined in the following.

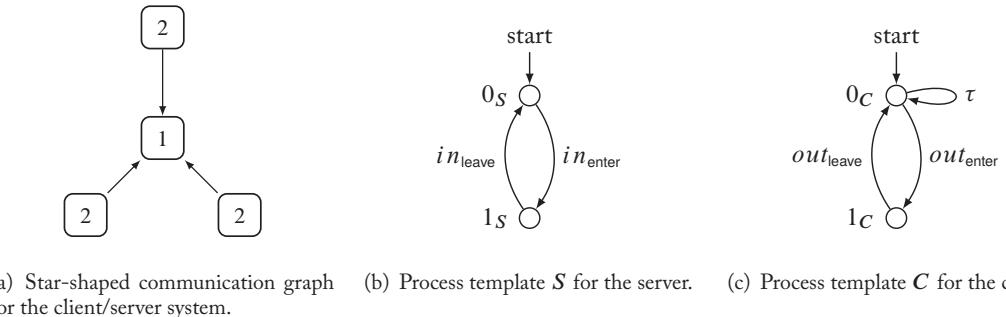


Figure 2.1: Communication graph and process templates for simple client/server system. Circles represent local states of a process, labeled with state names.

Synchronization constraint card. A *synchronization constraint* is a set $\text{card} \subseteq \mathbb{N}_0$ of natural numbers. This set is used to define the number of processes that participate in a synchronized action: The idea is that if a process with index v takes an out_a action then simultaneously some subset of v 's recipients take in_a actions. The cardinality of this subset must be in card . See Section 2.2.1 for synchronization cardinalities that correspond to standard synchronization primitives such as pairwise rendezvous, asynchronous rendezvous, and broadcast, in which card are very simple sets such as $\{1\}$, $\{0, 1\}$ and \mathbb{N}_0 .

Example 2.2 Synchronization in Client/Server System. In our client/server system in Figure 2.1, consider the synchronization constraint $\{1\}$. Intuitively, this means that whenever a client

8 2. SYSTEM MODEL AND SPECIFICATION LANGUAGES

wants to take a synchronous transition, labeled out_{enter} or out_{leave} , it can only do so when there is at least one other process which synchronizes with the client by taking a transition labeled with in_{enter} or in_{leave} , respectively. In the given communication graph, this other process can only be the server.

In the following definition of a system instance, we formalize the notion of synchronization.

SYSTEM INSTANCE \overline{P}^G

Given a system arity d , a d -ary system template \overline{P} , a d -ary connectivity graph $G = (V, E, \text{type})$ with $|V| = k$, and a synchronization constraint card, define the *system instance* $\text{sys}(\overline{P}, G, \text{card})$, also written \overline{P}^G if card is clear, as the *finite LTS* $(S, S_0, \Sigma_{\text{int}} \cup \Sigma_{\text{sync}}, \Delta, \Lambda)$ over indexed atomic propositions $\text{AP}_{\text{pr}} \times [k]$, where the following holds true.

- The state set S consists of all $|V|$ -tuples $s = (q_1, \dots, q_{|V|}) \in Q^{\text{type}(1)} \times \dots \times Q^{\text{type}(|V|)}$. For $v \in V$, let $s(v)$ denote q_v , the local state of the process at v . Each s is called a *global state*.
- The set of *global initial states* is defined as $S_0 := Q_0^{\text{type}(1)} \times \dots \times Q_0^{\text{type}(|V|)}$.
- The *global transition relation* $\Delta \subseteq S \times (\Sigma_{\text{int}} \cup \Sigma_{\text{sync}}) \times S$ is defined as the set of all internal transitions (in which a single process acts) and synchronous transitions (in which some set of processes act simultaneously, where we have the following.
 - An *internal transition* is an element (s, a, s') of $S \times \Sigma_{\text{int}} \times S$ for which there exists a process index $v \in V$ satisfying the following two conditions:
 - (int-step)** $s(v) \xrightarrow{a} s'(v)$ is a local transition of process template $P^{\text{type}(v)}$.
 - (int-frame)** For all $w \in V \setminus \{v\}$, $s(w) = s'(w)$.
 - A *synchronous transition* is an element (s, a, s') of $S \times \Sigma_{\text{sync}} \times S$ for which there exists a process index $v \in V$, called the *initiator*, and a set $\mathcal{I} \subseteq \{w \in V : E(v, w)\}$ of recipients of v in G such that:
 - (card)** $|\mathcal{I}| \in \text{card}$.
 - (step)** $s(v) \xrightarrow{out_a} s'(v)$ is a local transition of process template $P^{\text{type}(v)}$.
 - (I-step)** For every $w \in \mathcal{I}$, $s(w) \xrightarrow{in_a} s'(w)$ is a local transition of process template $P^{\text{type}(w)}$.
 - (frame)** For every $w \in V \setminus (\mathcal{I} \cup \{v\})$, $s'(w) = s(w)$.
 - (max)** There does not exist a strict superset $\mathcal{I}' \supset \mathcal{I}$ of recipients of v in G such that
 - (i) $|\mathcal{I}'| \in \text{card}$, and (ii) for all $w \in \mathcal{I}'$ there exists $r \in Q^{\text{type}(w)}$ such that $s(w) \xrightarrow{in_a} r$ is a local transition of process template $P^{\text{type}(w)}$.
- The labeling $\Lambda(s) \subseteq \text{AP}_{\text{pr}} \times [k]$ for $s \in S$ is defined as follows: for $v \in V$, $p_v \in \Lambda(s)$ if and only if $p \in \lambda^{\text{type}(v)}(s(v))$, and for $v \notin V$, $p_v \notin \Lambda(s)$ for all $s \in S$.

2.2.1 SOME STANDARD SYNCHRONIZATION PRIMITIVES

We give various synchronization constraints card that model standard forms of communication, namely pairwise rendezvous, asynchronous rendezvous, and broadcast (Chapter 5). Variations of the primitives, along with extensions of our basic system model, are used to define the Token-Passing Systems (Chapter 4), Guarded Protocols (Chapter 6), and Ad-Hoc Networks (Chapter 7); see Figure 2.2.

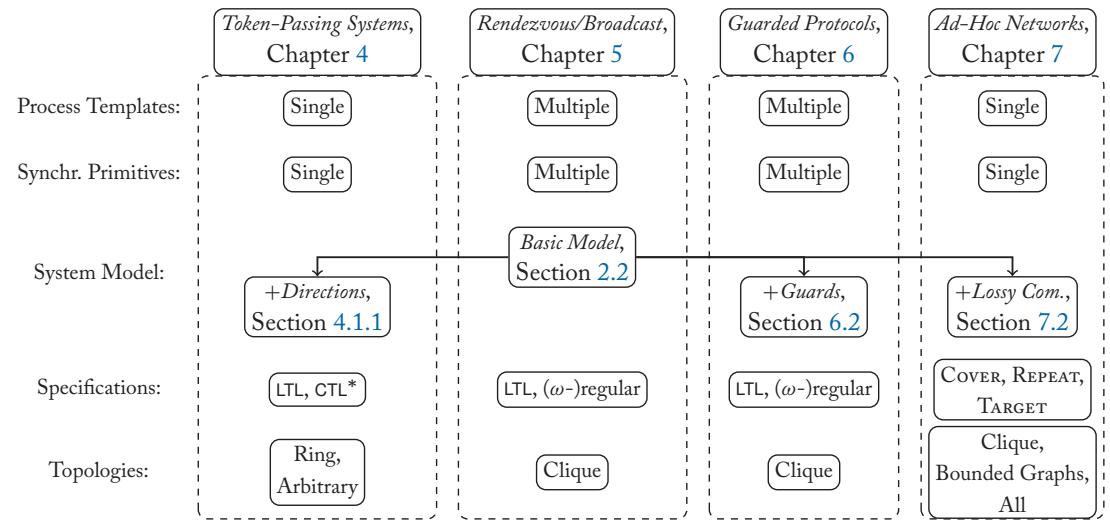


Figure 2.2: Basic system model and extensions in this survey. For topologies, “All” refers to the parameterized topology with all possible topologies in it, while “Arbitrary” refers to an arbitrary parameterized topology.

Pairwise Rendezvous. The synchronization constraint $\text{card} = \{1\}$ captures pairwise rendezvous—i.e., synchronization between pairs of processes. In this case it is customary to write $a!$ instead of out_a and $a?$ instead of in_a .

To illustrate, we instantiate the definition of synchronous transition for $\text{card} = \{1\}$. *Pairwise-rendezvous transitions* are of the form (s, a, s') for which there exists $(v, w) \in E$ such that:

(STEP) $(s(v), a!, s'(v))$ is a local transition of process template $P^{\text{type}(v)}$;

(I-STEP) $(s(w), a?, s'(w))$ is a local transition of process template $P^{\text{type}(w)}$; and

(FRAME) for all $z \notin \{v, w\}$, $s(z) = s'(z)$.

Note that this incorporates the (CARD) condition with $\mathcal{I} = \{w\}$, and the (MAX) condition is trivially satisfied since $\mathcal{I}' \supset \{w\}$ implies $|\mathcal{I}'| \notin \text{card}$.

10 2. SYSTEM MODEL AND SPECIFICATION LANGUAGES

Figure 2.3(a) illustrates some possible configurations for a pairwise-rendezvous step.

Asynchronous Rendezvous. The synchronization constraint $\text{card} = \{0, 1\}$ captures asynchronous rendezvous. The idea is that a process can synchronize with zero or one other process. The maximality condition (**MAX**) ensures that asynchronous rendezvous is like pairwise rendezvous except that a process taking a transition labeled out_a is not blocked when there are no corresponding processes that are able to take a transition labeled in_a .

Figure 2.3(b) illustrates some possible configurations for an asynchronous-rendezvous step.

Broadcast. The synchronization constraint $\text{card} = \mathbb{N}_0$ captures broadcast. The idea is that if vertex v wants to broadcast message a then every recipient of v that is ready to receive the broadcast message a does so. If no process is ready to receive the broadcast then since $0 \in \text{card}$, the initiating process makes its local transition anyway. For broadcast, out_a is written $a!!$ and in_a is written $a??$.

Again, we instantiate the definition of synchronous transition to illustrate. *Broadcast transitions* are of the form (s, a, s') where there exists $v \in V$ and a subset $\mathcal{I} \subseteq V$ of recipients of v , such that (STEP) $s(v) \xrightarrow{a!!} s'(v)$ is a local transition of $P^{\text{type}(v)}$, (\mathcal{I} -STEP) for all $w \in \mathcal{I}$, $s(w) \xrightarrow{a??} s'(w)$ is a local transition of $P^{\text{type}(w)}$, (FRAME) for all other z , $s'(z) = s(z)$, and (**MAX**) there is no recipient w of v not already in \mathcal{I} such that $s(w) \xrightarrow{a??} r$ is a local transition of $P^{\text{type}(w)}$ for some state r .

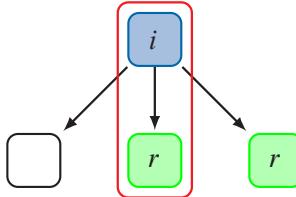
Thus, the (**MAX**) condition ensures that when a process v takes a transition labeled $a!!$ then every recipient w of v that can take a local transition labeled $a??$ does so. As a special case we get the broadcast protocols of [Esparza et al. \[1999, Section 2.1\]](#) in which “a process is always willing to receive a broadcast message.”

Figure 2.3(c) illustrates some possible configurations for a broadcast step.

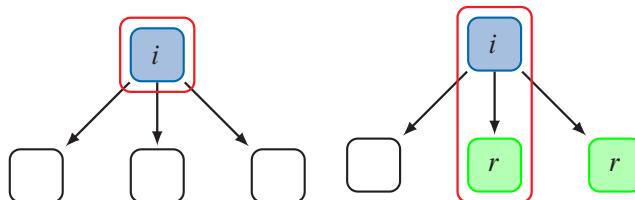
Variations on standard primitives. To model token passing systems, in Chapter 4 we introduce a variation of pairwise-rendezvous synchronization which keeps track of who has the token, and only allows this process to initiate a synchronization. To model synchronization failures in ad hoc networks, in Chapter 7 we introduce a *lossy* variant of broadcast synchronization by removing condition (**MAX**) from the definition of synchronous transition. Informally, lossy broadcast is to broadcast what asynchronous rendezvous is to pairwise rendezvous.

Figure 2.3(d) illustrates some possible configurations for a lossy broadcast step.

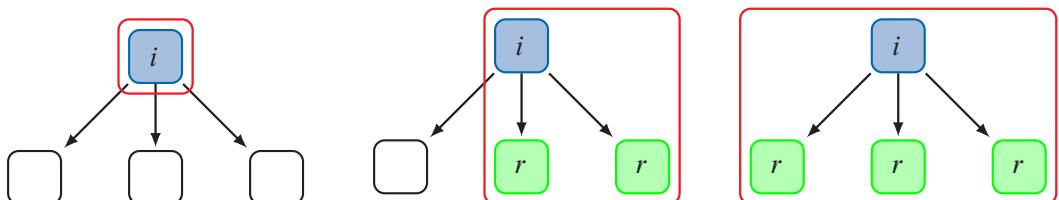
Notation. Note that, in general, the definition of a process template, including its transition labels, does not determine a synchronization primitive. This is highlighted in Examples 2.1 and 2.2, where the transition systems for server and client use the generic notation for actions in_{enter} , in_{leave} , out_{enter} , out_{leave} , and one can consider systems that use these process templates, but differ in the synchronization primitive.



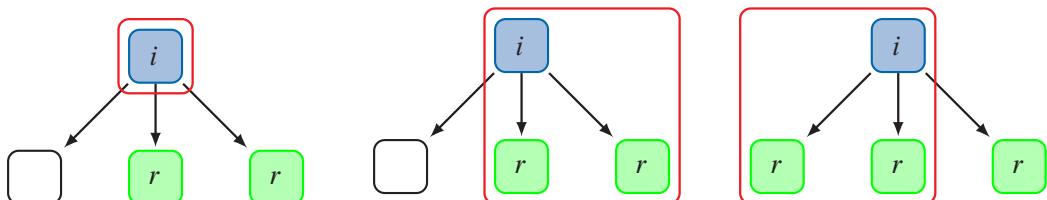
(a) **Pairwise rendezvous:** synchronization with exactly one recipient. If more than one process is ready to synchronize, only one of them can take the synchronous transition.



(b) **Asynchronous rendezvous:** synchronization with at most one recipient. If no process is ready, the initiator can take the transition on its own. Otherwise, exactly one of the recipients synchronizes.



(c) **Broadcast:** synchronization with all recipients that are ready. If no process is ready, the initiator can take the transition on its own. Otherwise, all recipients that are ready synchronize.



(d) **Lossy broadcast:** synchronization with some of the recipients that are ready. Even if other processes are ready, the initiator can also take the transition on its own.

Figure 2.3: Synchronization primitives: pictures show initiator of a synchronous transition, marked with i , with all its recipients. Processes that are ready to take the receive action are marked with r . The processes in the box take the synchronous transition.

12 2. SYSTEM MODEL AND SPECIFICATION LANGUAGES

In the following, when we use the special notation of the form $a!, a?$ for pairwise-rendezvous actions, or $a!!, a??$ for broadcast actions in a process template, then we implicitly fix the synchronization primitive to correspond to these action labels.

Systems with multiple primitives $\text{sys}(\overline{P}, G, \overline{\text{card}})$. By introducing a little more notation into Definition 2.2 we may define a parameterized system that allows more than one synchronization primitive. In this case, we have a tuple of synchronization constraints $\overline{\text{card}} = (\text{card}_1, \dots, \text{card}_k)$ and an equal number of sets of synchronizing-action labels $\Sigma_{\text{card}_i} \subset \Sigma_{\text{sync}}$. Then define card_i -*synchronous transition* as in the definition of synchronous transition but replace Σ_{sync} with Σ_{card_i} . Now define Δ to be the union of the internal transitions and all the card_i -synchronous transitions. Write $\text{sys}(\overline{P}, G, \overline{\text{card}})$ for the resulting system instance.

2.2.2 RUNS AND DEADLOCKS

A state s of an LTS M is *deadlocked* if there is no transition with source s . Given that the synchronization primitives discussed above are non-trivial, deadlocks may occur in a system instance. The existence of deadlocks leads to two problems. First, a deadlock is usually considered as an error that we would like to detect. Second, a deadlocked run is *finite*, and the specifications we consider are usually interpreted over *infinite* runs (see Section 2.4). Therefore, we need special ways to interpret runs that end in a deadlock.

Deadlock detection. Deadlock detection, i.e., checking whether deadlocked states are reachable, is a problem on its own, and solutions depend heavily on the underlying synchronization primitives. We are interested in *parameterized deadlock detection*, i.e., checking if there exists an instance of a parameterized system in which a deadlocked state is reachable. For some of the systems that we consider, certain assumptions guarantee that there can never be deadlocks. This is the case for token-passing systems without direction-awareness (cf. Section 4). In a few cases, deadlocks are possible and there exist decidability results for parameterized deadlock detection. For example, some of the cutoffs for parameterized model checking of guarded protocols in Section 6 are also cutoffs for the parameterized deadlock detection problem. In most of the cases, however, there are no known results for parameterized deadlock detection.

Interpreting deadlocked runs. The literature we survey here has three ways to deal with deadlocked runs, i.e., finite paths (starting in initial states) which end in deadlocked states.

1. Define a run to be an infinite path starting in the initial state. Thus, we evaluate specifications over infinite paths only and ignore maximal finite paths. This is the approach taken by German and Sistla [1992].
2. Define a run to be a maximal (finite or infinite) path that starts in the initial state. This requires one to use a specification language that can also be interpreted over finite paths. This is the approach taken by Emerson and Namjoshi [1995, 2003], Baier and Katoen [2008], and Delzanno et al. [2010].

3. Change the LTS M to get an LTS M' that does not contain deadlocked states by (i) introducing a new symbol τ into the set of action labels and (ii) for every deadlocked global state s add the transition (s, τ, s) . Evaluate specifications over infinite paths of M' that start in an initial state. This is the approach taken by Clarke et al. [1999].

Throughout this book we will explain in every chapter how deadlocked runs are interpreted, and mention known results for the parameterized deadlock detection problem.

2.3 PARAMETERIZED FAMILY OF UNIFORM CONCURRENT SYSTEMS

A *parameterized d-ary connectivity graph* is a sequence \mathbf{G} of d-ary connectivity graphs. Observe that as a special case, $\mathbf{G}(n)$ may contain n vertices and $d = 1$, e.g., $\mathbf{G}(n)$ may be the ring of size n . In general $\mathbf{G}(n)$ need not have n vertices.

Fix a d-ary system template \overline{P} , a parameterized d-ary connectivity graph \mathbf{G} , and a synchronization constraint card. These determine a *parameterized family of uniform concurrent systems* or simply a *parameterized system*, defined as the sequence

$$n \mapsto \text{sys}(\overline{P}, \mathbf{G}(n), \text{card}).$$

Notation $\overline{P}^{\mathbf{G}(n)}$. As a shorthand we may write $\overline{P}^{\mathbf{G}(n)}$ instead of $\text{sys}(\overline{P}, \mathbf{G}(n), \text{card})$ if card is clear from the context.

2.4 PARAMETERIZED SPECIFICATIONS

Specifications express the required behavior of systems. Typical specifications for concurrent systems are that no bad *global* state is ever reached (safety specifications), and that individual processes make progress (liveness specifications). Both kinds of specifications are naturally expressed by quantifying over the processes of the system: the canonical approach is to index processes as well as atomic propositions, so that one can express, e.g., that process i is in the critical section by “critical $_i$,” and further one can express that some process is in the critical section by $\bigvee_{i \in [n]} \text{critical}_i$. Observe that this Boolean formula is actually parameterized by n , and further that the formula—and the set of atomic propositions—grows with n . Using indexed propositions and parameterized Boolean operations in temporal logic, we obtain *indexed temporal logics*. In this book, instead of parameterized Boolean operations we will use the more intuitive notation of quantification over indices. That is, $\bigvee_{i \in [n]} \text{critical}_i$ will be expressed as $\exists i. \text{critical}_i$.

Apart from being a natural formalism, the question arises whether it is *necessary* to resort to a logic that induces an unbounded number of atomic propositions to express properties of parameterized systems. To answer this question, first consider the mutual exclusion property, stating that two distinct processes are never in the critical section at the same time, i.e., $\mathbb{G}(\forall i, j, i \neq j. (\neg \text{critical}_i \vee \neg \text{critical}_j))$. This formula belongs to the interesting fragment of in-

14 2. SYSTEM MODEL AND SPECIFICATION LANGUAGES

dexed temporal logic in which temporal operators do not appear inside the scope of index quantifiers. If a given specification is in this fragment, we can also express it in (non-indexed) temporal logic, for a modified system that can be obtained from the original one by introducing additional atomic propositions. For this example, introduce a proposition p and define a labeling function that maps a global state s to p if at most one process is in the critical section in s . (Browne et al. [1989] made this observation for the property that *exactly* one process is in the critical section.) Then, mutual exclusion is expressed as Gp , which is a formula in (non-indexed) temporal logic. Other specifications in this fragment can be found in distributed algorithms. Examples are the consensus problem [Lynch, 1996], where several processes need to agree on a common output after limited communication, or the reliable broadcast specification [Srikanth and Toueg, 1987], which ensures that all processes deliver the same set of messages.

However, one can show that even if we allow to redefine systems in such a way, non-indexed temporal logic is not sufficiently expressive for other specifications: for instance, consider the property that any process that tries to enter the critical section will eventually enter it, i.e., $\forall i. G(\text{trying}_i \rightarrow (\text{Fcritical}_i))$. Note that in this case, quantified subformulas contain temporal operators. To express this specification, we need to distinguish global states according to the processes that are in specific local states (e.g., trying or critical) in order to evaluate that some process actually makes progress along a path of a computation. As the number of processes is a priori not fixed in parameterized model checking, a fixed finite set of atomic propositions is not sufficient for this purpose. We are hence forced to use a logic where the number of propositions grows with the system size, and indexed temporal logic is a very natural choice for the formalization of such specifications.

2.4.1 INDEXED TEMPORAL LOGICS

Indexed temporal logics were introduced by Browne et al. [1989] to model specifications of certain concurrent systems. They are obtained by adding *index quantifiers* to a given temporal logic over indexed atomic propositions.

For example, we can specify a mutual exclusion property as $G(\forall i, j : i \neq j. (\neg\text{critical}_i \vee \neg\text{critical}_j))$. Furthermore, in a uni-directional ring, we can express that if a property p holds for some process, then p' should hold for its successor in the ring by writing $\forall i, j : j \in E(i). p_i \rightarrow p'_j$.¹

In the following, we first recall the syntax and semantics of CTL*. Then we extend CTL* to *indexed* CTL*. By restricting the temporal logic one gets fragments such as *indexed LTL*\X. All variations of indexed temporal logics from the PMC literature are fragments of indexed CTL*.

CTL*

We briefly describe syntax and semantics of CTL* (see, for instance, Clarke et al. [1999], and Baier and Katoen [2008]).

¹Emerson and Namjoshi [1995, 2003] write such formulas as $\forall i. p_i \rightarrow p'_{i+1}$ where addition is modulo the size of the ring.

Syntax. The syntax of CTL^* over a set AP of atomic propositions is defined as follows.

State formulas are formed according to the grammar

$$\Phi ::= \top \mid p \mid \Phi_1 \wedge \Phi_2 \mid \neg \Phi \mid E\varphi,$$

where $p \in \text{AP}$, Φ_1 and Φ_2 are state formulas and φ is a path formula.

Path formulas are formed according to the grammar

$$\varphi ::= \Phi \mid \varphi_1 \wedge \varphi_2 \mid \neg \varphi \mid X\varphi \mid \varphi_1 \cup \varphi_2,$$

where Φ is a state formula and $\varphi, \varphi_1, \varphi_2$ are path formulas. From these we derive, as usual, Boolean operators (like \vee), temporal operators F (eventually) and G (always), and the universal path quantifier A .

Semantics. Formulas are interpreted in labeled transition systems $M = (Q, Q_0, \Sigma, \delta, \lambda)$ over atomic propositions AP . For a state s of M , define

- $M, s \models p$ iff $p \in \lambda(s)$,
- $M, s \models \Phi_1 \wedge \Phi_2$ iff $M, s \models \Phi_1$ and $M, s \models \Phi_2$,
- $M, s \models \neg \Phi$ iff not $M, s \models \Phi$, and
- $M, s \models E\varphi$ iff $M, \pi \models \varphi$ for some infinite path π of M starting in s .

For an infinite path π of M starting in s , write π^j for suffix of π starting at position $j \geq 0$, and define

- $M, \pi \models \Phi$ iff $M, s \models \Phi$,
- $M, \pi \models \varphi_1 \wedge \varphi_2$ iff $M, \pi \models \varphi_1$ and $M, \pi \models \varphi_2$,
- $M, \pi \models \neg \varphi$ iff not $M, \pi \models \varphi$,
- $M, \pi \models X\varphi$ iff $M, \pi^1 \models \varphi$, and
- $M, \pi \models \varphi_1 \cup \varphi_2$ iff $\exists j \geq 0 \ \forall k < j. (M, \pi^j \models \varphi_2 \text{ and } M, \pi^k \models \varphi_1)$.

Finally, define $M \models \Phi$ if for all initial states s of M , it holds that $M, s \models \Phi$.

Note that action labels Σ are not used in the semantics.

Fragments. An important fragment of CTL^* is LTL , which consists of all CTL^* formulas that start with a universal path quantifier A , followed by a CTL^* formula without path quantifiers E or A . If it is clear that a formula is in LTL , then the leading path quantifier A may be omitted. From CTL^* and LTL , we obtain fragments $\text{CTL}^* \setminus X$ and $\text{LTL} \setminus X$, respectively, by only considering formulas that do not contain the next-state operator X .

16 2. SYSTEM MODEL AND SPECIFICATION LANGUAGES

Indexed CTL*

Fix an infinite set $I = \{i, j, \dots\}$ of index variables.

Syntax. The syntax of indexed CTL* over a set AP of atomic propositions and set I of index variables is defined as follows.

State formulas are formed according to the grammar

$$\Phi ::= \top \mid p_i \mid \forall i : r. \Phi \mid \exists i : r. \Phi \mid \Phi_1 \wedge \Phi_2 \mid \neg \Phi \mid E\varphi,$$

where $p \in AP$, $i \in I$, Φ, Φ_1, Φ_2 are state formulas, φ is a path formula, and r is a list of *range expressions* from

$$\text{type}(i) = t \mid neq(i_1, \dots, i_n) \mid i \in E(j),$$

where $t \in \mathbb{N}$ and $i, i_1, \dots, i_n, j \in I$.

Path formulas are formed according to the grammar

$$\varphi ::= \Phi \mid \forall i : r. \varphi \mid \exists i : r. \varphi \mid \varphi_1 \wedge \varphi_2 \mid \neg \varphi \mid X\varphi \mid \varphi_1 \cup \varphi_2$$

where Φ is a state formula and $\varphi, \varphi_1, \varphi_2$ are path formulas.

Our notation is inspired by that of Emerson and Namjoshi [1995, 2003]. The symbols \forall and \exists are called *index quantifiers*. If r is empty, we just write $\forall i. \phi$ or $\exists i. \phi$. The expression $neq(i, j)$ may also be written $i \neq j$. Variable i and atoms p_i in an indexed CTL* formula are *bound* if they are in the scope of a quantifier $\forall i : r$ or $\exists i : r$. An indexed CTL* formula Φ is a *sentence* if every atom in it is bound. An indexed CTL* formula ϕ in which index variables i_1, \dots, i_k are not bound by index quantifiers may be written $\phi(i_1, \dots, i_k)$. We say that i_1, \dots, i_k are *free* in ϕ .

Semantics. An indexed CTL* formula ϕ is interpreted over a system instance \overline{P}^G (with \overline{P} a system template and $G = (V, E, \text{type})$ a connectivity graph) and a *valuation of the index variables* $e : I \rightarrow V$.

A *valuation e* satisfies the range expression $\text{type}(i) = t$ if $\text{type}(e(i)) = t$. Similarly, it satisfies $neq(i_1, \dots, i_n)$ if $e(i_1), \dots, e(i_n)$ are pairwise distinct, and it satisfies $i \in E(j)$ if $(e(i), e(j)) \in E$. Finally, it satisfies a list of range expressions if it satisfies every element of the list. A valuation $e : I \rightarrow V$ is an *i-variant* of a valuation e' if $e'(j) = e(j)$ for all $j \in I \setminus \{i\}$.

Define $\overline{P}^G, e, s \models \Phi$ inductively:

- $\overline{P}^G, e, s \models p_i$ iff $p_{e(i)} \in \Lambda(s)$,
- $\overline{P}^G, e, s \models \forall i : r. \Phi$ iff for all *i*-variants e' of e that satisfy r it holds that $\overline{P}^G, e', s \models \Phi$,

- $\overline{P}^G, e, s \models \exists i : r. \Phi$ iff there exists an i -variant e' of e that satisfies r and it holds that $\overline{P}^G, e', s \models \Phi$,
- $\overline{P}^G, e, s \models \Phi_1 \wedge \Phi_2$ iff $\overline{P}^G, e, s \models \Phi_1$ and $\overline{P}^G, e, s \models \Phi_2$,
- $\overline{P}^G, e, s \models \neg\Phi$ iff not $\overline{P}^G, e, s \models \Phi$, and
- $\overline{P}^G, e, s \models E\varphi$ iff $\overline{P}^G, e, \pi \models \varphi$ for some infinite path π of \overline{P}^G starting in s .

For an infinite path π of \overline{P}^G starting in s , define

- $\overline{P}^G, e, \pi \models \Phi$ iff $\overline{P}^G, e, s \models \Phi$,
- $\overline{P}^G, e, \pi \models \forall i : r. \varphi$ iff for all i -variants e' of e that satisfy r it holds that $\overline{P}^G, e', \pi \models \varphi$,
- $\overline{P}^G, e, \pi \models \exists i : r. \varphi$ iff there exists an i -variant e' of e that satisfies r and it holds that $\overline{P}^G, e', \pi \models \varphi$,
- $\overline{P}^G, e, \pi \models \varphi_1 \wedge \varphi_2$ iff $\overline{P}^G, e, \pi \models \varphi_1$ and $\overline{P}^G, e, \pi \models \varphi_2$,
- $\overline{P}^G, e, \pi \models \neg\varphi$ iff not $\overline{P}^G, e, \pi \models \varphi$,
- $\overline{P}^G, e, \pi \models X\varphi$ iff $\overline{P}^G, e, \pi^1 \models \varphi$, and
- $\overline{P}^G, e, \pi \models \varphi_1 \cup \varphi_2$ iff $\exists j \geq 0 \ \forall k < j. (\overline{P}^G, e, \pi^j \models \varphi_2 \text{ and } \overline{P}^G, e, \pi^k \models \varphi_1)$.

If Φ is a sentence, define $\overline{P}^G, s \models \Phi$ if for all evaluations (equivalently, for some evaluation) $e : I \rightarrow V$ it holds that $\overline{P}^G, e, s \models \Phi$. Finally we define:

$$\overline{P}^G \models \Phi$$

if for every initial state s of \overline{P}^G it holds that $\overline{P}^G, s \models \Phi$.

Notation. We write $P^G, s \models \Phi(c_1, \dots, c_k)$ if for all evaluations e in which i_j maps to c_j (for $j \leq k$) it holds that $P^G, e, s \models \Phi(i_1, \dots, i_k)$.

Fragments. A *prenex* formula is of the form $Q_1 i_1, \dots, Q_k i_k : r. \phi(i_1, \dots, i_k)$, where the Q_j are index quantifiers and $\phi(i_1, \dots, i_k)$ has no index quantifiers. Formulas of the form $Q_1 i_1, \dots, Q_k i_k : r. \phi(i_1, \dots, i_k)$, for some range expression r , will be referred to as *k-indexed*.²

²Note that in these formulas only the last quantifier has a range expression. This is not an additional restriction, however, since semantically $Q_1 i_1 : r_1 \dots Q_k i_k : r_k. \phi(i_1, \dots, i_k)$ with separate range expressions r_1, \dots, r_k is equivalent to $Q_1 i_1, \dots, Q_k i_k : r_1 \wedge \dots \wedge r_k. \phi(i_1, \dots, i_k)$ with the conjunction of all range expressions.

18 2. SYSTEM MODEL AND SPECIFICATION LANGUAGES

Since the results in the literature concentrate on the prenex fragment of indexed-temporal logic, we introduce some shorthands for important fragments. Write ICTL^* for the set of all prenex indexed CTL^* sentences, and $k\text{-CTL}^*$ for the set of all k -indexed formulas in ICTL^* .

Prenex indexed LTL, written ILTL , is defined as the linear-time fragment of ICTL^* , i.e., formulas of the form $Q_1 i_1, \dots, Q_k i_k : r. A\phi(i_1, \dots, i_k)$, where the Q_j are index quantifiers and $\phi(i_1, \dots, i_k)$ has no index quantifiers and no path quantifiers.³ Write $k\text{-LTL}$ for the set of all k -indexed formulas in ILTL .

Fragments of ICTL^* without the next operator, such as $\text{ICTL}^* \setminus X$ and $\text{ILTL} \setminus X$, are obtained by disallowing formulas that contain the X operator. Specifications without the next-time operator X are common in the literature, since they are stuttering-insensitive, and thus are a natural specification language for asynchronous concurrent systems. Moreover, the presence of X , even with weak communication primitives, leads to undecidability of the PMCP [Emerson and Kahlon, 2003b].

2.4.2 ACTION-BASED SPECIFICATIONS

In contrast to specifications in index temporal logic, which are formulas over indexed state labels, an action-based specification evaluates the sequences of action labels of \overline{P}^G . An ω -regular action-based specification is an ω -regular language \mathcal{L} over actions $\Sigma_{int} \cup \Sigma_{sync}$.

It is interpreted as follows: write $\overline{P}^G \models \mathcal{L}$ if the projection onto $\Sigma_{int} \cup \Sigma_{sync}$ of every infinite path of \overline{P}^G starting in an initial state is in the language \mathcal{L} .

We also find regular action-based specifications which are (ordinary finite-word) regular languages \mathcal{L} over actions $\Sigma_{int} \cup \Sigma_{sync}$. These are interpreted as follows: $\overline{P}^G \models \mathcal{L}$ if every finite prefix of every projection onto $\Sigma_{int} \cup \Sigma_{sync}$ of every path of \overline{P}^G starting in an initial state is in the language \mathcal{L} .

We write $\text{Reg}(A)$ and $\omega\text{Reg}(A)$ for regular and ω -regular action-based specifications, respectively.

2.4.3 SPECIFICATIONS IN THE LITERATURE

Most of the decidability results in the literature are based on one of the following forms of specifications.

- Plain temporal logics (or ω -regular properties) over state labels of a specific process template [Emerson and Kahlon, 2003b, Emerson and Namjoshi, 1996, 1998, German and Sistla, 1992]. These are used for systems with a special *control* process and an unbounded number of *user* processes, and specify only the behavior of the controller.

³Note that the index variables are bound *outside* of the temporal path quantifier. In particular, for an existentially quantified formula to be satisfied there must exist a valuation of the variables such that ϕ holds for all paths (and not one valuation for each path).

- Indexed temporal logics over the state labels of all processes [Clarke et al., 2004, Emerson and Kahlon, 2000, 2003a,c, 2004, Emerson and Namjoshi, 1995, 2003, German and Sistla, 1992]. These are used for systems with or without a unique control process.
- Action-based specifications on global transitions [Emerson and Kahlon, 2003b, Emerson and Namjoshi, 1998, Esparza et al., 1999]. Specifications on the actions of the system do not directly talk about the state of local processes. However, as argued by Esparza et al. [1999], in many cases one can construct a modified system that satisfies an adapted specification over state labels if and only if the original system satisfies the original action-based specification.

In addition, we consider special parameterized model checking problems COVER (reachability of a local state in one of the components), REPEAT (repeated reachability of a local state in the same component), and TARGET (reachability of a certain state in all components simultaneously, not definable in prenex ICTL*) [Abdulla et al., 2013a, Delzanno et al., 2010, 2011, 2012b]. These are used when reasoning about ad-hoc networks with an arbitrary (and possibly changing) graph structure, and constitute small and important fragments that may be decidable even for systems where more general languages are undecidable (see Chapter 7).

2.5 MODEL CHECKING PROBLEMS FOR CONCURRENT SYSTEMS

Typical questions from the PMC literature are of the form: Is the PMCP decidable for a given class of systems (defined by system template, connectivity graph, and synchronization constraint) and a given class of specifications? We formalize such questions in the following. Fix:

- a set of system templates \mathcal{P} (such as the d-tuples of finite process templates),
- a parameterized connectivity graph \mathbf{G} (such as the sequence of rings),
- a tuple of synchronization constraints $\overline{\text{card}}$ (which is often a singleton card), and
- a set of indexed-temporal logic or action-based specifications \mathcal{F} (such as k-CTL*).

First, we define the (non-parameterized) model checking problem for our class of systems:

The model-checking problem MCP($\mathcal{P}, \mathbf{G}, \mathcal{F}, \overline{\text{card}}$) is as follows:

Input. $\overline{P} \in \mathcal{P}, \phi \in \mathcal{F}$ and $n \in \mathbb{N}$.

Output. “Yes” if $\text{sys}(\overline{P}, \mathbf{G}(n), \overline{\text{card}}) \models \phi$; and “No” otherwise.

20 2. SYSTEM MODEL AND SPECIFICATION LANGUAGES

Unless explicitly stated otherwise, in this book we assume that all system instances $\text{sys}(\overline{P}, \mathbf{G}(n), \text{card})$ are finite-state, and thus that the (non-parameterized) model checking problem is decidable.

In contrast to this, in the parameterized model checking problem we do not give n as an input, but ask whether the specification holds for *all* n :

The parameterized model-checking problem $\text{PMCP}(\mathcal{P}, \mathbf{G}, \mathcal{F}, \overline{\text{card}})$ is as follows:

Input. $\overline{P} \in \mathcal{P}$, and $\phi \in \mathcal{F}$.

Output. “Yes” if $\text{sys}(\overline{P}, \mathbf{G}(n), \overline{\text{card}}) \models \phi$ for all $n \in \mathbb{N}$; and “No” otherwise.

When $\overline{\text{card}}$ is clear from the context, we write $\text{PMCP}(\mathcal{P}, \mathbf{G}, \mathcal{F})$.

Example 2.3 Consider the question: Is the PMCP decidable for token-passing systems on unidirectional rings, with specifications in $\text{ILTL} \setminus X$? This problem is considered in Section 4.2.1 and written $\text{PMCP}(\mathcal{P}_{\text{simptok}}, \mathbf{R}, \text{ILTL} \setminus X, \{1\})$ where $\mathcal{P}_{\text{simptok}}$ is a set of processes that are designed to simulate token-passing using pairwise rendezvous, i.e., $\text{card} = \{1\}$, and $\mathbf{R}(n)$ is the uni-directional ring of size n .

2.5.1 COMPUTABILITY ASSUMPTIONS

In parameterized model checking one usually makes the following very general assumptions. In the literature they are typically not made explicit.

- Each process template P is computable, meaning there is an algorithm that computes exactly the states and transitions of P .
- Each set $\text{card} \subseteq \mathbb{N}_0$ is computable.
- The parameterized connectivity graph is computable, meaning that there is an algorithm that on input $n \in \mathbb{N}$ outputs a list of the vertices and edges in $\mathbf{G}(n)$.

Note that for the systems under consideration in this book, the first two are satisfied by definition, since P is finite-state, and card is finite or co-finite for the standard communication primitives. Furthermore, note that the first assumption is also satisfied by various classes of infinite-state processes that are considered in the parameterized verification literature, such as pushdown machines or processes with variables that range over the natural numbers. From these assumptions we can conclude that:

- (i) the inputs to the PMCP are finite objects, e.g., \overline{P} may be finite state, or (the configuration space of) a counter machine; and

- (ii) the parameterized system is computable, meaning that there is an algorithm that given n returns a finitary representation of the system instance $\text{sys}(\overline{P}, \mathbf{G}(n), \text{card})$, e.g., a list of the states and transitions, or in case the system instance is infinite it may be an algorithm that computes the states and transitions.

The first item is needed for the PMCP to be an algorithmic problem, and the second item is needed if there is to be any hope of deciding the PMCP (for particular cases).

CHAPTER 3

Standard Proof Machinery

The purpose of this section is to briefly summarize the typical principles and ingredients in proofs of (un)decidability for the PMC problem.

3.1 TECHNIQUES TO PROVE UNDECIDABILITY OF PMCP

To prove that $\text{PMCP}(\mathcal{P}, \mathbf{G}, \mathcal{F})$ is undecidable it is sufficient to computably transform a given Turing Machine T into a tuple $(\bar{\mathcal{P}}, \mathbf{G}, \phi) \in \mathcal{P} \times \mathcal{G} \times \mathcal{F}$ such that T does not halt on empty input if and only if for all $n \in \mathbb{N}$, $\bar{\mathcal{P}}^{\mathbf{G}(n)} \models \phi$.

The first undecidability proof that applies to uniform parameterized systems is by [Suzuki \[1988\]](#), for systems consisting of identical processes arranged in a uni-directional ring with a single multi-valued token. The proof reduces non-halting of Turing machines to this problem. A neater exposition by [Emerson and Namjoshi \[1995\]](#) goes via two-counter machines. Informally, a two-counter machine [[Minsky, 1967](#)] has a read-only tape, finite control, and two counters, with operations to increment a counter, decrement a counter, test a counter for zero, and change its internal state. It accepts the word on its input tape if there is a run starting with zero in both of its counters that leads to a halting state. Two-counter machines are Turing-complete. Obviously, k -counter machines for $k > 2$ are also Turing-complete.

We use a formal definition of counter machine close to the one given by [Emerson and Kahlon \[2003b\]](#). Although it may look more complicated than the usual Minsky machines, we picked their definition for the purpose of presentation, as it typically makes PMCP undecidability proofs easier.

Counter Machines. An *input-free k -counter machine* (k CM) \mathcal{M} consists of a set of locations $[m]$ (for some $m \in \mathbb{N}$), and action set $\mathcal{A} = \{inc(c_q), dec(c_q), zero(c_q): q \in [k]\}$, and a set $\Delta \subseteq [m] \times \mathcal{A} \times [m]$ of *commands*. A *configuration* of \mathcal{M} is a tuple $(i, \bar{v}) \in [m] \times \mathbb{N}_0^k$; so i is a location and v_j indicates the value of the j -th counter. The *initial configuration* is $(1, 0, \dots, 0)$; so the machine starts in location 1 with empty counters. The *halting configurations* are $\{m\} \times \mathbb{N}_0^k$; so the machine halts whenever it reaches location m). The *configuration graph* of \mathcal{M} is the directed graph whose vertices are all the configurations, and edges are defined as follows. There is an edge from configuration (i, \bar{v}) to configuration (j, \bar{w}) if (a) $i \neq m$ and (b) there exists a command $(i, \sigma, j) \in \Delta$ such that:

- if σ is $inc(c_q)$ then $w_q = v_q + 1$ and $w_p = v_p$ for $p \neq q$;

24 3. STANDARD PROOF MACHINERY

- if σ is $dec(c_q)$ then $w_q = v_q - 1$ and $w_p = v_p$ for $p \neq q$; and
- if σ is $zero(c_q)$ then $w_q = v_q = 0$ and $w_p = v_p$ for $p \neq q$.

Note that if σ is $dec(c_q)$ then the described edge exists only if $v_q > 0$ (i.e., there is an implicit test for non-zero).

This definition differs from Minsky's original definition of multi-counter machine in two ways: (i) a Minsky machine is deterministic while this definition is non-deterministic and (ii) a Minsky machine has no deadlocked configurations while this definition allows non-halting configurations with no outgoing edge. The translation of a Minsky machine into our definition results in a machine such that for every location $l \in [m]$ either there is exactly one outgoing transition, and it is labeled $inc(c_q)$ for some q , or there are exactly two outgoing transitions and they are labeled $dec(c_q)$ and $zero(c_q)$ for some q . From this translation and undecidability of the halting problem for Minsky machines [Minsky, 1967], the following theorem is immediate.

Theorem 3.1 *The following problem is undecidable: given an input-free k -counter machine \mathcal{M} (for $k \geq 2$), where the counters are initialized with zeroes, does there exist a path in the configuration space of \mathcal{M} from the initial configuration to a halting configuration?*

Undecidability of PMCP. Thus, typical undecidability proofs in this area show how to simulate a 2CM by the composition of (uniform) finite-state processes. Typically one process, the *controller* process, simulates the internal state of the 2CM, while the remaining n *storage* processes collectively encode the counter values (the simulation either covers n steps of the 2CM, or a prefix of the computation as long as the counter values are bounded by n). The work in the proof is typically showing how the controller can issue commands to increment/decrement counters and test counters for zero. The prototypical such proof, following Emerson and Namjoshi [1995, 2003], is sketched in Section 4.2.3.

3.2 HOW TO PROVE DECIDABILITY OF THE PMCP

In this section we describe techniques that help one prove that the PMCP is decidable, i.e., symmetry arguments, reductions to well-structured transition systems or vector addition systems, and cutoff techniques.

A natural first step of a decidability proof is to use symmetry of the systems \overline{P}^G as well as the specifications to simplify the problem. For instance:

- if the connectivity graph G is a clique, and all processes are instances of the same process template P , then $P^G \models \forall i.\phi(i)$ is equivalent to $P^G \models \phi(1)$, or
- as noted by Emerson and Namjoshi [1995, 2003], if the connectivity graph G is a ring, and all processes are instances of the same process template P , then $P^G \models \forall i, j : i \neq j.\phi(i, j)$ is equivalent to $P^G \models \forall j.\phi(1, j)$.

Such intuitions can be formalized following the approach by [Emerson and Sistla \[1996\]](#). Formal treatment of symmetry is tedious and thus often only addressed implicitly in the literature (and this survey). Note that [Emerson and Namjoshi \[1995, 2003\]](#) treated symmetry explicitly.

We can reduce the PMCP to model checking a single infinite-state LTSs by combining \overline{P}^G for $G \in \mathbf{G}$. Model checking is decidable for certain classes of infinite-state systems and certain specifications. Notably, the coverability problem (defined below) is decidable for well-structured transition systems (WSTS); see [Finkel and Schnoebelen \[2001\]](#) or [Abdulla et al. \[1996\]](#). Vector addition systems with states (or equivalently Petri Nets) are special kinds of WSTSs for which repeated coverability and reachability problems are decidable. For classical results in Petri nets, see [Esparza \[1998\]](#) and [Yen \[1992\]](#). Thus, for example, if the combined system is a WSTS or a VASS, then we can decide the PMCP. We review the definitions and decidability results for WSTS and VASS in Section 3.2.1.

Another approach is to reduce PMCP to model checking finitely many system instances, see e.g., [Emerson and Namjoshi \[1995, 2003\]](#) and [Clarke et al. \[2004\]](#). Such results are often described as cutoffs and are typically proved by exhibiting suitable simulation relations between almost all system instances of the parameterized system and a fixed system instance. We formalize this in Section 3.2.3.

3.2.1 WELL-STRUCTURED TRANSITION SYSTEMS

Following the work by [Abdulla et al. \[1996\]](#) and [Finkel and Schnoebelen \[2001\]](#), well-structured transition systems became a popular framework for reasoning about infinite-state systems and parameterized systems. We begin by recalling some required notions.

A binary relation \leq on a set X is a *quasi-order* if it is reflexive and transitive. For $T \subseteq X$ write $\uparrow T := \{x \in X \mid \exists t \in T, t \leq x\}$, called the *upward closure* of T . The *downward closure* of T is the set $\{x \in X \mid \exists t \in T, x \leq t\}$. A set T is *upward closed* if $T = \uparrow T$.

A quasi-order is a *well quasi-order* if for every infinite sequence x_1, x_2, \dots there exist indices $i < j$ with $x_i \leq x_j$. In particular, there are no infinite decreasing chains and no infinite anti-chains.

A typical example of a well quasi-order on \mathbb{N}^n (for $n \in \mathbb{N}$) is defined by $(x_1, \dots, x_n) \leq (y_1, \dots, y_n)$ if for every $i \leq n$ it holds that $x_i \leq y_i$.

The following properties follow from the definition of well quasi-order \leq . If C is upward closed then there is a finite set $B \subseteq C$, called a *basis* of C , such that C is the upward closure of B : $C = \uparrow B$ (the word *basis* is used to mean that B generates C , and not that B is unique with this property). If $C_1 \subseteq C_2 \subseteq C_3 \subseteq \dots$ is an infinite chain of upward closed sets then there exists k such that $C_k = \cup_{n \in \mathbb{N}} C_n$.

Let \leq be a quasi-order on the set of states of a transition system M . For states s, t of M , write $s \xrightarrow{a} t$ if there is an edge in M from s to t labeled a , write $s \rightarrow t$ if $s \xrightarrow{a} t$ for some a , and write $s \xrightarrow{*} t$ if there is a path in M from s to t . For a set C of states of M , define the set $Pred(C)$ of states M as follows: $s \in Pred(C)$ iff either $s \in C$ or there exists $t \in C$ such that

26 3. STANDARD PROOF MACHINERY

$s \rightarrow t$. Say that \leq is *monotonic* with respect to M if for all states s, s', t of M and all action labels a , if $s \leq s'$ and $s \xrightarrow{a} t$, then there exists $t' \geq t$ such that $s' \xrightarrow{a} t'$.¹ The following property holds if \leq is monotonic: if C is an upward-closed set of states of M , then $Pred(C)$ is upward closed.

Definition 3.2 Let M be a transition system and let \leq be an ordering on the state set of M . Then M is a *well-structured transition system* (WSTS) with respect to \leq if the following conditions hold:

1. \leq is a well-quasi ordering,
2. \leq is monotonic with respect to M , and
3. one can compute, from a basis for an upward closed set C , a basis for $Pred(C)$.

Finkel and Schnoebelen [2001] gave a detailed discussion of fundamental computation models that are well-structured transition systems. Examples are variants of, e.g., communication finite state machines, basic process algebra, and Petri nets.

The following problem is parameterized by an infinite-state LTS M with initial states I and a quasi-ordering \leq on the states S of M .

- The *coverability* problem:
input: finite set of states $T \subset S$.
output: ‘Yes’ if and only if there exist $i \in I$ and $t \in \uparrow T$ such that $i \xrightarrow{*} t$.

The following theorem can be easily obtained from the work by Abdulla et al. [1996] and Finkel and Schnoebelen [2001].

Theorem 3.3 *The coverability problem is decidable for WSTSs, if the initial state set I satisfies one of the following conditions.*

1. I is finite.
2. I is downward closed, and has a computable set membership.

Proof. The proof idea is similar in both cases. From a basis for C and n , one can compute a basis for $Pred^n(C)$. Thus, one can compute, given n , whether $Pred^n(C) = Pred^{n+1}(C)$: they are equal iff for every s' in a basis of $Pred^{n+1}(C)$ there is s in a basis of $Pred^n(C)$ such that $s \leq s'$. Thus, one can compute a number k such that $Pred^k(C) = \bigcup_{n \in \mathbb{N}} Pred^n(C)$, and hence a basis for $\bigcup_{n \in \mathbb{N}} Pred^n(C)$.

Now consider the coverability problem for state set T and initial state set I . Let B be a basis for $\bigcup_{n \in \mathbb{N}} Pred^n(\uparrow T)$. There exists $i \in I$ and $t \in \uparrow T$ such that $i \xrightarrow{*} t$ if and only if $I \cap \uparrow B \neq \emptyset$.

¹Monotonicity is called “strong compatibility” by Finkel and Schnoebelen [2001].

In other words, we have reduced the coverability problem to checking if there is some $i \in I$ and some $b \in B$ such that $b \leq i$. If I is finite then there are only finitely many pairs to check. If I is downward closed then $I \cap \uparrow B \neq \emptyset$ is equivalent to $I \cap B \neq \emptyset$. If I also has decidable membership then one simply needs to check if one of the finitely many elements of B is in I . \square

This theorem is used to prove decidability of PMCP for safety properties in Theorem 5.7, Theorem 6.31, and Theorem 7.19.

3.2.2 VECTOR ADDITION SYSTEMS WITH STATES (AND PETRI NETS)

We define a *vector addition system with states* (VASS) M to consist of a finite set Q of states, an initial state $\iota \in Q$, a finite action set $A \subset \mathbb{Z}^d$, and finite transition relation $T \subseteq Q \times A \times Q$. The *configuration space* of a VASS M is the transition system with state set $Q \times \mathbb{N}_0^d$, initial state $(\iota, 0, \dots, 0)$, and transitions defined by $(q, \bar{x}) \xrightarrow{a} (r, \bar{y})$ if $\bar{y} = \bar{x} + a$ where $(q, a, r) \in T$. Write $s \xrightarrow{*} t$ if there is a path in the configuration space (ignoring actions) from s to t .

The VASS with ordering \leq defined as follows is a WSTS: $(q, \bar{x}) \leq (r, \bar{y})$ if $q = r$ and $x_i \leq y_i$ for $1 \leq i \leq d$. Besides the coverability problem, we also consider the following.

- The *reachability* problem:
input: configuration $t = (q, \bar{v}) \in Q \times \mathbb{N}_0^d$.
output: ‘Yes’ if and only if $\iota \xrightarrow{*} t$ for some $\iota \in I$.
- The *control-state repeated coverability* problem:
input: $q \in Q$.
output: ‘Yes’ if and only if there exist infinitely many configurations t_1, t_2, \dots each of whose first coordinate is q and such that $(\iota, 0, \dots, 0) \xrightarrow{*} t_1$ and $t_i \xrightarrow{*} t_{i+1}$ for all $i \in \mathbb{N}$.

The following theorem summarizes results on Petri nets and VASS important for the exposition in this book, cf. [Esparza, 1998] for a detailed survey of this area.

Theorem 3.4 *The control state repeated coverability problem for VASS is EXPSPACE-complete. Moreover, the problem can be solved in space which is polynomial in the number of states of the VASS and exponential in the dimension of the VASS. The reachability problem is decidable (and EXPSPACE-hard).*

This theorem is used to prove decidability of special PMCPs of broadcast and pairwise systems for liveness properties in Theorem 5.8, Theorem 5.10, and Theorem 7.21.

A VASS in which $|Q| = 1$ is called a vector-addition system (VAS), which is a notational variant of Petri nets. Conversely, every VASS can be represented as a VAS (since the state component can be coded in $|Q|$ many additional coordinates).

28 3. STANDARD PROOF MACHINERY

3.2.3 DECOMPOSITIONS AND CUTOFFS

Some proofs that $\text{PMCP}(\mathcal{P}, \mathbf{G}, \mathcal{F})$ is decidable also yield a computable reduction to a finite set of finite-state model checking problems. This is formalized as follows.

A *decomposition* for $(\mathcal{P}, \mathbf{G}, \mathcal{F})$ is a function mapping every pair $(\bar{P}, \phi) \in \mathcal{P} \times \mathcal{F}$ (where \bar{P} and \mathbf{G} have the same arity d) to a pair $(\text{MC}_{fin}, \Phi_{\mathbb{B}})$, where

1. MC_{fin} is a set $\{(K_1, \phi_1), \dots, (K_m, \phi_m)\}$, where each K_i is a d -ary connectivity-graph, each ϕ_i is a formula, m is a positive integer, and
2. $\Phi_{\mathbb{B}}$ is a Boolean formula $\Phi_{\mathbb{B}}(x_1, \dots, x_m)$ (with m variables)

such that the following are equivalent:

- $\forall n \in \mathbb{N}, \bar{P}^{\mathbf{G}(n)} \models \phi$,
- $\Phi_{\mathbb{B}}(c_1, \dots, c_m)$ evaluates to \top under the assignment $c_i = \top$ iff $\bar{P}^{K_i} \models \phi_i$.

Thus, a decomposition reduces the problem $\text{PMCP}(\mathcal{P}, \mathbf{G}, \mathcal{F})$ to a Boolean combination of the finitely many model checking problems from the set MC_{fin} . Note that if the decomposition is a computable function, then $\text{PMCP}(\mathcal{P}, \mathbf{G}, \mathcal{F})$ is decidable.

We identify a special case of this definition that appears in the literature: we say m is a *cutoff* (for $(\mathcal{P}, \mathbf{G}, \mathcal{F})$) if for all $(\bar{P}, \phi) \in \mathcal{P} \times \mathcal{F}$, the image of the decomposition function is the pair $(\text{MC}_{fin}, \Phi_{\mathbb{B}})$ where $\text{MC}_{fin} = \{(\mathbf{G}(1), \phi), \dots, (\mathbf{G}(m), \phi)\}$ and $\Phi_{\mathbb{B}} = x_1 \wedge \dots \wedge x_m$. In other words, m is a cutoff means that for all $(\bar{P}, \phi) \in \mathcal{P} \times \mathcal{F}$: $\forall n, \bar{P}^{\mathbf{G}(n)} \models \phi$ if and only if $\forall n \leq m, \bar{P}^{\mathbf{G}(n)} \models \phi$. Cutoff results appear in Section 4 and Section 6.

Proposition 3.5 *If $(\mathcal{P}, \mathbf{G}, \mathcal{F})$ has a cutoff, then $\text{PMCP}(\mathcal{P}, \mathbf{G}, \mathcal{F})$ is decidable.*

Proof. Suppose m is a cutoff. This means that for all $(\bar{P}, \phi) \in \mathcal{P} \times \mathcal{F}$: $\forall n, \bar{P}^{\mathbf{G}(n)} \models \phi$ if and only if $\forall n \leq m, \bar{P}^{\mathbf{G}(n)} \models \phi$. Thus, an algorithm solving the PMCP is: given $\bar{P} \in \mathcal{P}, \phi \in \mathcal{F}$, output “Yes” if for every $n \leq m$, it holds that $\bar{P}^{\mathbf{G}(n)} \models \phi$, and “No” otherwise. \square

Remark 3.6 To solve PMCP, it is not enough to know that there exists a cutoff, one also needs to know the value of the cutoff in order to know when to stop enumerating the system instances.

For instance, suppose that for every $k \in \mathbb{N}$ the data $(\mathcal{P}, \mathbf{G}, k\text{-CTL}^*)$ has a cutoff. This means that for every k there is an algorithm deciding $\text{PMCP}(\mathcal{P}, \mathbf{G}, k\text{-CTL}^*)$. However, it does not imply that there is an algorithm that can decide $\text{PMCP}(\mathcal{P}, \mathbf{G}, \text{ICTL}^*)$. However, if the cutoffs are computable (i.e., there is an algorithm that given $k \in \mathbb{N}$ outputs a cutoff for $(\mathcal{P}, \mathbf{G}, k\text{-CTL}^*)$), then $\text{PMCP}(\mathcal{P}, \mathbf{G}, \text{ICTL}^*)$ is decidable. Such a situation occurs in Section 4.

Cutoff results for $(\mathcal{P}, \mathbf{G}, \mathcal{F})$ are often obtained by showing that for any $\bar{P} \in \mathcal{P}$, the set of runs of all system instances of $\bar{P}^{\mathbf{G}}$ cannot be distinguished by any specification in \mathcal{F} from the

set of runs of the system instance $\overline{P}^{\mathbf{G}(m)}$, for some $m \in \mathbb{N}$. This is the case if one can show, for instance, that (i) for sufficiently many components, adding an extra component to the system only adds runs that can already be simulated in the smaller system, and (ii) a run in a system instance with many components can be simulated by a run in a system with a smaller number of components. This is often possible when the connectivity-graph is “homogeneous” (like a star or a clique), see [Emerson and Kahlon \[2000\]](#) and [German and Sistla \[1992\]](#). For the case of unidirectional rings and specifications in indexed CTL* $\backslash X$, [Emerson and Namjoshi \[1995, 2003\]](#) use stuttering bisimulations (see, e.g., [Browne et al. \[1988\]](#)) between tuples of processes in system instances of different size. To prove that two system instances are bisimilar, one has to find one or several maps (depending on the number of indexed variables) from components of the big system to components of the small system, such that any projection of a path of the big system with respect to these maps is stuttering equivalent to a path of the small system, and vice versa.

CHAPTER 4

Token-passing Systems

In a token-passing system (TPS), processes communicate by passing a token to their neighbors in the connectivity graph. We will define the passing of a token (with or without a value) as a special case of pairwise-rendezvous synchronization. In contrast to most of the other classes of systems in this survey, TPSs have been analyzed on complex connectivity graphs, where connections may or may not be labeled with directions. Thus, in order to define TPSs we extend our basic system model to *direction-aware parameterized systems*.

One special case has received attention in the literature, in particular in the work by Emerson and Namjoshi [1995, 2003] and Clarke et al. [2004] and recently by Aminof et al. [2014a]: connections in the graphs are not labeled with directions and the token does not hold a value. In this case, the only purpose of the token is to distinguish the process that currently holds it from all the other processes. This allows us to realize systems with mutual exclusion properties as TPSs, where the process with the token is the only one to access the shared resource. Connectivity graphs can then be used to model certain scheduling, e.g., token rings model a round-robin scheduling scheme.

Environment assumptions can be added to the specification in order to further restrict token passing, e.g., to only consider runs with fair token passing. Emerson and Namjoshi [1995, 2003] show that in this case the PMCP is decidable when the considered graphs are rings and specifications are in a fragment of $\text{CTL}^*\backslash\chi$ with purely universal index quantification, and Clarke et al. [2004] show decidability for arbitrary graphs and specifications in $k\text{-LTL}\backslash\chi$ (for any fixed k). Recently, these two decidability results have been unified and extended by Aminof et al. [2014a].

We also consider systems where the token can hold different values, and, in later sections, look at systems where processes are aware of direction labels in the connectivity graph. While it is known that multi-valued tokens lead to an undecidable PMCP in general [Suzuki, 1988], there are decidability results for systems with multi-valued tokens and direction-awareness, subject to additional restrictions on the process and the specifications.

Running Example. As the running example of this chapter, we use a version of Milner's scheduler [Milner, 1989]. The scheduler is composed of an arbitrary number of components, each of which is responsible for activating one (unspecified) task and receives confirmation when its task has been executed. Scheduling should ensure that the tasks are activated in a round-robin scheme, and that every task has terminated before being activated again. This can naturally be modeled in token-passing systems, as noted by Emerson and Namjoshi [1995, 2003].

4.1 SYSTEM MODEL

To model TPSs, we first extend our general system model to *direction-aware parameterized systems*. Then we define TPSs as a special case, where token passing is modeled by restricted pairwise-rendezvous synchronization. Not all results of this chapter require this extension of the system model: results in Section 4.2 consider direction-unaware systems, while results in Section 4.3 consider direction-aware systems.

4.1.1 DIRECTION-AWARE PARAMETERIZED SYSTEMS

We extend the definition of parameterized systems to include additional labels on edges, called *directions*. The idea is that processes are allowed to restrict, depending on their local state, which edges of the connectivity graph can be used to synchronize with other processes.

A *direction-aware parameterized system* consists of the same ingredients as a (direction-unaware) parameterized system (see Section 2.2), except for the following modifications.

Directions. Fix finite sets Dir_{out}, Dir_{in} of *outgoing* and *incoming directions*.

Directed connectivity graph. A *d-ary directed connectivity graph* is a tuple $G = (V, E, \text{type}, dir_{out}, dir_{in})$, where (V, E, type) is a d-ary connectivity graph, and $dir_{out} : E \rightarrow Dir_{out}$ and $dir_{in} : E \rightarrow Dir_{in}$ are additional labeling functions. A *parameterized d-ary directed connectivity graph* is a sequence \mathbf{G} of d-ary directed connectivity graphs.

Example 4.1 Bi-directional Ring. Let $Dir_{out} = Dir_{in} = \{\text{cw}, \text{ccw}\}$, standing for clockwise and counterclockwise directions in a ring. A *bi-directional ring* is a directional connectivity graph $B = (V, E, \text{type}, dir_{out}, dir_{in})$ with

- $V = [k]$ for some $k \geq 2$,
- $E = \{(i, i +_k 1), (i, i -_k 1) \mid i \in V\}$, and
- $dir_{out}(i, i +_k 1) = dir_{in}(i, i +_k 1) = \text{cw}$, and $dir_{out}(i, i -_k 1) = dir_{in}(i, i -_k 1) = \text{ccw}$ for $i \in V$.

In general, type is arbitrary, but usually we consider bi-directional rings with a single process template.

A *parameterized bi-directional ring* \mathbf{B} is a sequence of B_1, B_2, \dots of bi-directional rings, usually with $V(B_n) = [n]$. A bi-directional ring with $V = [4]$ is depicted in Figure 4.1.

Direction-aware Transition Labels. In a direction-aware parameterized system, process templates use transition labels from

$$\Sigma_{pr} := \Sigma_{int} \cup \{(out_a, d) : a \in \Sigma_{sync}, d \in Dir_{out}\} \cup \{(in_a, d) : a \in \Sigma_{sync}, d \in Dir_{in}\}.$$

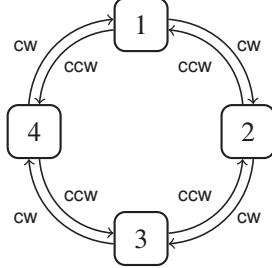


Figure 4.1: A bi-directional ring with 4 nodes. Since $dir_{out}(e) = dir_{in}(e)$ for all edges e , the direction is only displayed once.

Direction-aware Synchronous Transitions. Internal transitions remain as before, but synchronous transitions are now direction-aware. That is, a synchronous transition must satisfy the same properties as before, and additionally the direction d in the local transition label of the sender must match the labeling of the used edge by dir_{out} , and similarly with dir_{in} for the receiver(s).

Formally, *direction-aware synchronous transitions* are elements (s, t, s') of $S \times \Sigma_{sync} \times S$ for which there exists a process index $v \in V$ and a set $\mathcal{I} \subseteq \{w \in V : E(v, w)\}$ of recipients of v in G such that:¹

(CARD) $|\mathcal{I}| \in \text{card.}$

(STEP) $s(v) \xrightarrow{o_{out}, d} s'(v)$ is a local transition of $P^{\text{type}(v)}$.

(DIR) For every $w \in \mathcal{I}$, $dir_{out}(v, w) = d$.

(\mathcal{I} -STEP-DIR) For every $w \in \mathcal{I}$, $s(w) \xrightarrow{i_{in}, d_w} s'(w)$ is a local transition of $P^{\text{type}(w)}$, with $d_w = dir_{in}(v, w)$.

(FRAME) For every $w' \in V \setminus \{v, w\}$, $s'(w') = s(w')$.

(MAX) There does not exist a strict superset $\mathcal{I}' \supset \mathcal{I}$ of recipients of v in G such that the conditions above are satisfied, with \mathcal{I}' replacing \mathcal{I} , and s'' replacing s' .

Example 4.2 Directional Pairwise Rendezvous. In a bi-directional ring, pairwise rendezvous as defined in Section 2.2.1 allows any process to synchronize with either of its two neighbors. We can define *directional pairwise rendezvous*, that may restrict which of its neighbors a process is allowed to synchronize with.

¹Note that (CARD), (FRAME), and (MAX) are as before.

34 4. TOKEN-PASSING SYSTEMS

To obtain directional rendezvous, we instantiate the definition of directional synchronous transition for $\text{card} = \{1\}$. *Directional pairwise-rendezvous transitions* are of the form (s, a, s') for which there exists $(v, w) \in E$ such that

(STEP) $s(v) \xrightarrow{a!,d} s'(v)$ is a local transition of $P^{\text{type}(v)}$.

(DIR) $\text{dir}_{out}(v, w) = d$.

(I-STEP-DIR) $s(w) \xrightarrow{a?,d_w} s'(w)$ is a local transition of $P^{\text{type}(w)}$, with $d_w = \text{dir}_{in}(v, w)$.

(FRAME) For every $w' \in V \setminus \{v, w\}$, $s'(w') = s(w')$.

As with non-directional pairwise rendezvous, this incorporates the (CARD) condition with $\mathcal{I} = \{w\}$, and the (MAX) condition is trivially satisfied since $\mathcal{I}' \supset \{w\}$ implies $|\mathcal{I}'| \notin \text{card}$.

Remark. Note that a direction-aware parameterized system degenerates to a parameterized system if $|\text{Dir}_{out}| = |\text{Dir}_{in}| = 1$. Also, by having either $|\text{Dir}_{out}| = 1$ or $|\text{Dir}_{in}| = 1$, we can define parameterized systems where processes can choose which edges to use for synchronization only for incoming or only for outgoing edges.

4.1.2 TOKEN-PASSING SYSTEMS

Fix a finite set T of token values. If $|\mathsf{T}| = 1$, we call the token a *simple token*, otherwise a *multi-valued token*.

Token-passing system template. For token-passing systems, we consider only 1-ary system templates, and therefore do not distinguish between process templates and system templates. A *token-passing process template* with token values T is a finite² LTS $P = (Q, Q_0, \Sigma_{pr}, \delta, \lambda)$ with the following restrictions.

1. The state set Q is partitioned into two non-empty sets: $Q = T \cup N$. States in T are said to *have the token*.
2. The initial state set is $Q_0 = \{\iota_T, \iota_N\}$ for some $\iota_T \in T, \iota_N \in N$.
3. The synchronous transition labels are given by the token values, i.e., $\Sigma_{sync} = \mathsf{T}$.
4. Every transition $q \xrightarrow{out_t,d} q'$ with $t \in \mathsf{T}$ satisfies that q has the token and q' does not. We say that P *sends the token* with value t in direction d .
5. Every transition $q \xrightarrow{in_t,d} q'$ with $t \in \mathsf{T}$ satisfies that q' has the token and q does not. We say that P *receives the token* with value t from direction d .

²Note that all results of this chapter that state existence of a cutoff hold even for infinite-state process templates.

6. Every transition $q \xrightarrow{a} q'$ with $a \in \Sigma_{int}$ satisfies that q has the token if and only if q' has the token.
7. Every infinite action-labeled run $a_0a_1\dots$ of P is in the set $(\Sigma_{int}^* Dir_{out} \Sigma_{int}^* Dir_{in})^\omega \cup (\Sigma_{int}^* Dir_{in} \Sigma_{int}^* Dir_{out})^\omega$. That is, the token is sent and received infinitely often in every run.

A More Liberal Token-passing Restriction. Restriction 6 above was introduced for direction-unaware systems by Emerson and Namjoshi [1995, 2003]. Aminof et al. [2014a] extended it to direction-aware systems, and showed that all the results in this chapter that state existence of cutoffs for TPSs also hold for the following, more liberal restriction:

- (†) From every state q that has the token there must be a path $q \dots q'$ such that q' does not have the token, and from every state q that does not have the token there must be a path $q \dots q'$ such that q' has the token.

Notation. Let \mathcal{P}_T be the set of all token-passing process templates with token values T . If $|T| = 1$ then we write \mathcal{P}_{simp} instead of \mathcal{P}_T . Moreover, let \mathcal{P}_T^u denote the set of all process templates for which $|Dir_{out}| = |Dir_{in}| = 1$, i.e., direction-unaware processes.³ If we require $|Dir_{in}| = 1$, then processes cannot choose from which directions to receive the token, but possibly in which direction to send it. Denote the set of all such process templates by \mathcal{P}_T^{snd} . Similarly, define \mathcal{P}_T^{rcv} to be all process templates where $|Dir_{out}| = 1$ — processes cannot choose where to send the token, but possibly from which direction to receive it.

Example 4.3 Processes in Milner’s Scheduler. Milner’s scheduler is composed of a variable number of processes, each responsible for scheduling one of the tasks. An implementation of such a process is depicted in Figure 4.2, where label A stands for activation of the task, and label C for completion of the task. Transitions to a state labeled with C are assumed to synchronize with actual completion of the task by an external device.

After activating the task, the process sends the token in direction cw , and afterwards waits for the return of the token, which may happen before or after completion of the task. Only after both of these events, the process will activate its task again.

Token-passing System Instance. To define token-passing systems, we instantiate the definition of a direction-aware parameterized system with

- $\Sigma_{sync} = T$,
- $d = 1$,

³For notational simplicity, we assume that Dir_{out} of the directed connectivity graph and the direction-aware process template are always identical, and similarly for Dir_{in} . One can easily define process templates to run on connectivity graphs with differing sets of directions, at the cost of notational overhead.

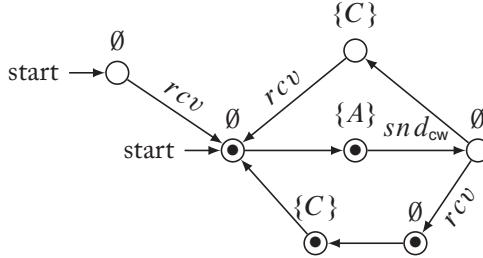


Figure 4.2: Process template for Milner’s scheduler. States with token are depicted with a dot inside the circle. State labels are depicted above states. Label A stands for activation of the task, C for completion of the task.

- $\text{card} = \{1\}$ (i.e., directional pairwise rendezvous), and
- process templates from \mathcal{P}_T .

Additionally, the global initial states S_0 are restricted to those states s for which there exists a unique $v \in V$ such that local state $s(v)$ has the token.⁴

Example 4.4 Milner’s Scheduler for 4 Tasks. We obtain an implementation of Milner’s scheduler for four tasks by constructing a system instance with four process templates as given in Figure 4.2, arranged in a bi-directional ring as given in Figure 4.1.

Note that since the processes always send the token in direction cw, their behavior would be the same in a uni-directional ring. Our system model allows us to define variants of this example where, e.g., the token is non-deterministically sent in one of the directions cw, ccw, and where the receiving process may accept the token from both directions, or only from one.

Deadlocks. Restrictions 6 and (\dagger) for token-passing process templates both guarantee the absence of deadlocks for direction-unaware systems. Note that the same is not true for direction-aware systems: the processes which are ready to receive might accept the token only from certain directions, which may be different from the directions into which the process that has the token can send it. This can lead to a deadlock if all process run into states from which the only possible transition is a token-passing transition. There are no known decidability results for parameterized deadlock detection in direction-aware TPSs.⁵

⁴Note that there is a slight difference between our model and that of Emerson and Namjoshi [1995, 2003]: in our model, one of the processes starts with the token, while in the original model nobody has the token, and the first transition of the system is a receive action of one process. There is, however, no difference in decidability of the two models, and the only difference in satisfaction of properties is in properties that mention the initial position of the token.

⁵Absence of deadlocks is guaranteed if we require that either (a) the process must always be able to reach a state such that it can receive the token from any direction, or (b) the process must always be able to reach a state such that it can send the token into any direction. Both (a) and (b) are more restrictive than (\dagger) , and incomparable to 6. The undecidability results in

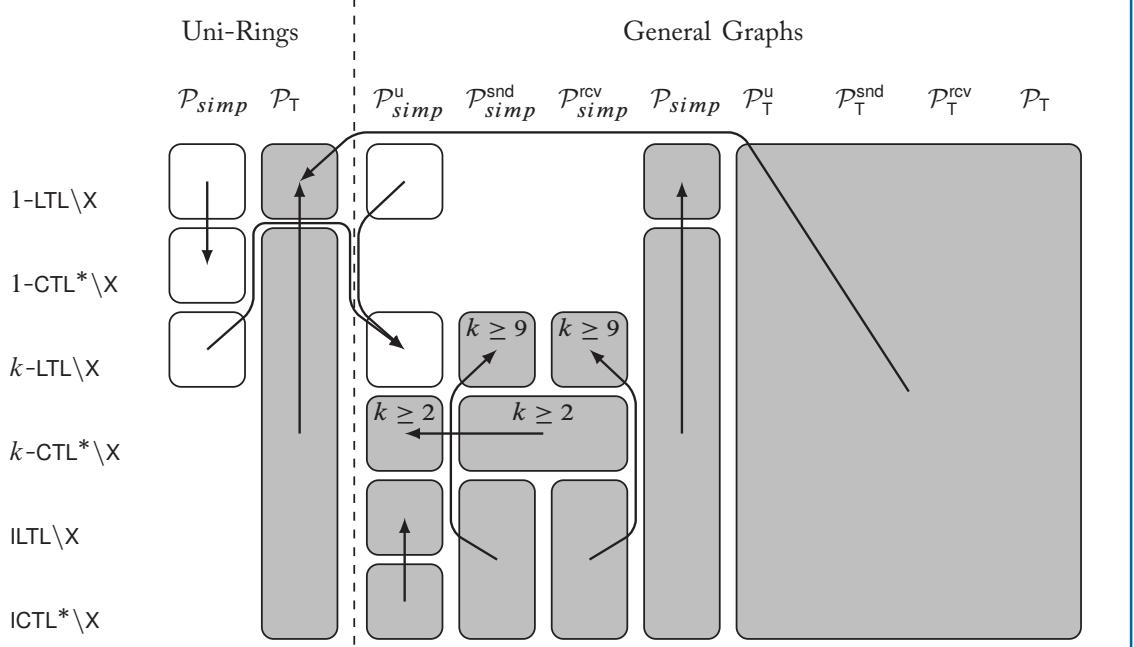


Figure 4.3: Decidability results and reductions for token-passing systems, distinguished into connectivity graphs and classes of process templates on top, and fragments of ICTL^* on the left. White boxes are decidability results, dark boxes are undecidability results. An arrow from box A to box B means that the result of A follows from the result, or a variation of the proof, of B . Blank items are not covered in the literature.

Specifications. For token-passing systems, we use indexed temporal logic specifications in fragments of $\text{ICTL}^* \setminus X$. Unless a different fragment is specified, in the following $\varphi(i_1, \dots, i_n)$ stands for a formula in $\text{ICTL}^* \setminus X$ with free index variables i_1, \dots, i_n .

4.2 RESULTS FOR DIRECTION-UNAWARE TOKEN-PASSING SYSTEMS

Figure 4.3 and Table 4.1 (on page 47) give an overview of the decidability results for TPSs. We consider results for direction-unaware systems (with process templates from \mathcal{P}_{simp}^u or \mathcal{P}_T^u) in this section, and for direction-aware systems (with process templates from \mathcal{P}_T^{snd} , \mathcal{P}_T^{rcv} , or \mathcal{P}_T) in Section 4.3.

Theorem 4.17 and the first part of Theorem 4.18 also hold for templates that satisfy restriction (a), while the second part of Theorem 4.18 also holds for templates that satisfy (b).

4.2.1 DECIDABILITY FOR SIMPLE TOKEN-PASSING IN UNI-DIRECTIONAL RINGS

Consider the parameterized connectivity graph \mathbf{R} of uni-directional rings of size n , i.e., $\mathbf{R}(n) = (V, E)$ with $V = [n]$ and $E = \{(i, j) : j = i + 1 \bmod n\}$. In a cornerstone paper, [Emerson and Namjoshi \[1995, 2003\]](#) prove that in uni-directional rings with a simple token it is enough to check a given property for all rings up to a given size, where the size depends on the quantification in the formula.

Theorem 4.5 [\[Emerson and Namjoshi, 2003\]](#). $\text{PMCP}(\mathcal{P}_{\text{simp}}^{\text{u}}, \mathbf{R}, \mathcal{F})$ has a cutoff if \mathcal{F} is one of the following fragments of $\text{CTL}^* \setminus \text{X}$:⁶

- (i) for $\mathcal{F} = \{\forall i. \varphi(i)\}_{\varphi(i) \in \text{CTL}^* \setminus \text{X}}$, the cutoff is 2.
- (ii) for $\mathcal{F} = \{\forall i, j : j \in E(i). \varphi(i, j)\}_{\varphi(i, j) \in \text{CTL}^* \setminus \text{X}}$, the cutoff is 3.
- (iii) for $\mathcal{F} = \{\forall i, j : i \neq j. \varphi(i, j)\}_{\varphi(i, j) \in \text{CTL}^* \setminus \text{X}}$, the cutoff is 4.
- (iv) for $\mathcal{F} = \{\forall i, j, l : \text{neq}(i, j, l), j \in E(i). \varphi(i, j, l)\}_{\varphi(i, j, l) \in \text{CTL}^* \setminus \text{X}}$, the cutoff is 5.

Proof idea. We first sketch the proof that 2 is a cutoff for formulas of the form $\forall i. \varphi(i)$, and then consider the case of multiple quantified index variables.

Consider formulas of the form $\forall i. \varphi(i)$. First, by symmetry of rings, a system instance $P^{\mathbf{R}(n)}$ in satisfies $\forall i. \varphi(i)$ if and only if it satisfies $\varphi(0)$. Second, consider system instances $P_0^{\mathbf{R}(n)}$, which are the same as $P^{\mathbf{R}(n)}$, except that the labeling function is restricted to atoms of the form p_0 for $p \in \text{AP}_{\text{pr}}$. Then there is a stuttering bisimulation between $P_0^{\mathbf{R}(n)}$ and $P_0^{\mathbf{R}(2)}$ for all $n \in \mathbb{N}$. A stuttering bisimulation implies that both system instances agree on all $\text{CTL}^* \setminus \text{X}$ formulas. The required bisimulation is defined as follows: two global states s, s' , in $P_0^{\mathbf{R}(n)}, P_0^{\mathbf{R}(n')}$ are equivalent iff $s(0) = s'(0)$, i.e., the processes with index 0 are in the same state, and thus either both have the token or both do not have the token. The key reasons why this is a stuttering bisimulation is that

- the processes with index 0 can mimick each other,
- the behavior of other processes can be ignored, except that they must allow the token to arrive at process 0 again, and
- a process $j \neq 0$ with the token can, by (\dagger) , send the token to its successor.

⁶In fact, the results of [Emerson and Namjoshi \[1995, 2003\]](#) hold for a logic that can also talk about local actions a_i of components. In the same vein, [Khalimov et al. \[2013a\]](#) proposed a local X operator over state labels as an extension of the logic that preserves the existence of cutoffs. None of the other results in this survey support such local properties, and therefore we have not included them in our definition in Section 2.4.1. We conjecture that most of the results in this chapter could be extended to such a logic.

Moreover, the labels of process 0 do not change during steps of the other processes, i.e., the system “stutters.”

For the case of multiple quantifiers, like $\forall i, j : i \neq j. \varphi(i, j)$, only the first quantifier can be instantiated with the fixed value 0, and for the second we need to consider all possible values for j in $[n] \setminus \{0\}$. In fact, we only need to consider the following cases:

- (i) j is recipient of i ,
- (ii) i is recipient of j , and
- (iii) there is no edge between i and j .

We define a monotone function $h : [n] \rightarrow [m]$, where m is the cutoff, such that each of (i), (ii), and (iii) holds for $(0, j)$ in $P^{\mathbf{R}(n)}$ if and only if it holds for $(0, h(j))$ in $P^{\mathbf{R}(m)}$. The function h exists if m is chosen big enough, for this type of specification $m = 4$ is sufficient. One can easily show that for every j , there is a stuttering bisimulation between processes $0, h(j)$ in $P^{\mathbf{R}(m)}$ and processes 0 and j in $P^{\mathbf{R}(n)}$. \square

Example 4.6 Verification of Milner’s Scheduler. As mentioned before, Milner’s scheduler is supposed to guarantee that all tasks are activated in a round-robin fashion, and that each task can terminate before being activated again (and only terminates after actually being activated). In Figure 4.2, activation of the task is represented by state label A , and termination by state label C . In a uni-directional ring, these requirements can then be expressed in the following parameterized specification:

$$\forall i, j, l : neq(i, j, l), j \in E(i). \text{AG} (A_i \rightarrow (\neg A_l \cup A_j))$$

$$\forall i. \text{AG} ((A_i \rightarrow (A_i \cup \neg A_i \cup C_i)) \wedge (C_i \rightarrow (C_i \cup \neg C_i \cup A_i))).$$

To verify this specification in systems with an arbitrary number of processes, it is sufficient to verify the first property in systems up to size 5 (by Theorem 4.5 (iv)), and the second property in systems up to size 2 (by Theorem 4.5 (i)). Thus, standard LTL model checking in rings of size up to 5 can be used to decide the PMCP for the given parameterized specification and implementation of Milner’s scheduler in uni-directional rings.

Extensions of Cutoff Results for Rings. Emerson and Namjoshi [1995, 2003] only provide cutoffs for properties with a small number k of index variables, but state that their technique is applicable for all k . Moreover, the original result is based on a strong restriction on process templates, stating that the sequence of actions from $\{in, out\}$ along every infinite path in P strictly alternates between in and out actions. We note that for their proof to work, restriction (†) above suffices.

40 4. TOKEN-PASSING SYSTEMS

Aminof et al. [2014a] investigated ways to compute cutoffs in general.

Theorem 4.7 [Aminof et al., 2014a]. *Let \mathcal{F}_k^* be the set of k -CTL* $\setminus X$ formulas with only universal or only existential index quantifiers. Then, for any given k , PMCP(\mathcal{P}_{simp}^u , \mathbf{R} , \mathcal{F}_k^*) has cutoff $2k$.⁷*

Proof idea. This result is obtained by applying the general proof technique we will see in the proof of Theorem 4.10 to the special case of uni-directional rings. In this case, we can construct the d -contractions for every k explicitly, and note that for every k and d , the d -contraction is a ring of size at most $2k$. \square

4.2.2 DECIDABILITY FOR SIMPLE TOKEN-PASSING IN GRAPHS

Some of the ideas for simple token-rings have been extended by Clarke et al. [2004] to simple token-passing systems on arbitrary classes of graphs. The main points of difference are: (i) properties are given in prenex $\text{ILTL} \setminus X$, and (ii) the decidability result is non-uniform and non-constructive. That is, the proof only shows that for every given k and parameterized graph \mathbf{G} , the PMCP for all formulas from k -LTL $\setminus X$ is decidable. The proof does not supply an effective way to construct this decision procedure, and also makes no statement about the decidability of the PMCP for the full logic $\text{ILTL} \setminus X$.

Theorem 4.8 [Clarke et al., 2004]. *PMCP(\mathcal{P}_{simp}^u , \mathbf{G} , k -LTL $\setminus X$) is decidable for every parameterized connectivity-graph \mathbf{G} and natural number k . Indeed, every $(\mathcal{P}_{simp}^u, \mathbf{G}, k\text{-LTL}\setminus X)$ has a decomposition.*

Proof idea. Recall that a decomposition is a mapping from inputs of the PMCP to a set of finite model checking problems MC_{fin} and a formula $\Phi_{\mathbb{B}}$ that combines the results of model checking (see Section 3.2). In this case, MC_{fin} is obtained by considering all so-called k -topologies of \mathbf{G} . These are graphs of size up to $2k$, obtained by fixing a set of k vertices in some $\mathbf{G}(n)$, and collapsing all other vertices into so-called hub vertices, such that indirect connections between the k fixed vertices are preserved.⁸ One can show that for every graph G , evaluating a k -LTL $\setminus X$ -formula ϕ in P^G amounts to a Boolean combination of the model-checking results of ϕ on the k -topologies of \mathbf{G} . Thus, the k -topologies, together with the original property, give the first part MC_{fin} of our decomposition. Since there are only finitely many k -topologies, and thus only finitely many Boolean formulas that combine these model checking results, one of them must be the $\Phi_{\mathbb{B}}$ that is the second part of the decomposition. \square

Looking closely at the decomposition, we note that the Boolean formula $\Phi_{\mathbb{B}}$ depends on the quantifier structure of ϕ as well as on \mathbf{G} . If there are no quantifier alternations, then $\Phi_{\mathbb{B}}$ is just

⁷Khalimov et al. [2013a, Corollary 2] stated that $2k$ is a cutoff for k -LTL $\setminus X$. Aminof et al. [2014a] noted that this is not the case for formulas with quantifier alternation.

⁸That is, there is a path between vertices v and v' that goes through a hub in the k -topology iff there is a path between v and v' that visits none of the k fixed vertices in the original graph. Note that this is a generalization of the idea by Emerson and Namjoshi [1995, 2003] that one of the main properties that needs to be preserved is whether or not vertices are neighbors.

a big conjunction or disjunction over all model-checking results for the k -topologies of \mathbf{G} , and thus we obtain an effective decision procedure whenever we can identify all of these k -topologies. If ϕ contains quantifier alternations, it is in general not known how to obtain $\Phi_{\mathbb{B}}$.⁹

Aminof et al. [2014a] looked at the question whether this proof can be made uniform for arbitrary parameterized graphs \mathbf{G} and specifications in $\text{ILTL} \setminus X$. They show that for certain parameterized connectivity graphs, the (uniform) PMCP is undecidable.

Theorem 4.9 [Aminof et al., 2014a]. *There exists a parameterized connectivity graph \mathbf{G} such that $\text{PMCP}(\mathcal{P}_{\text{simp}}^{\text{u}}, \mathbf{G}, \text{ILTL} \setminus X)$ is undecidable.*

Proof idea. Define P to be the process template with two states, one with and one without the token. Every transition must change the state, which, in particular, satisfies condition (\dagger) from the definition of TPSs in Section 4.1.2. For $i, j \in \mathbb{N}$ let $L_{i,j}$ be the lasso with prefix of length i , and a loop of length j . Based on some computable ordering on all deterministic Turing machines (for example, lexicographic), let \mathbf{G} consist of all graphs of the form $L_{i,j}$ such that the j th Turing machine halts on the empty word in at most i steps. Note that this is a computable set.

For every $k \in \mathbb{N}$, let φ_k be the formula (in $k\text{-LTL} \setminus X$)

$$\forall i_1, \dots, i_k : \text{neq}(i_1, \dots, i_k). \neg (\mathsf{F}(\text{tok}_{i_1} \cup \text{tok}_{i_2} \cup \dots \cup \text{tok}_{i_k} \cup \text{tok}_{i_1})) ,$$

where tok_i is a predicate which is true iff process i has the token. Thus, $P^G \models \varphi_k$ if and only if G does not contain a ring of size k as a subgraph.

Now reduce the non-halting problem for Turing Machines to the PMCP:

the k th TM does not halt

\Leftrightarrow \mathbf{G} does not contain a graph with a ring of size k as subgraph

$\Leftrightarrow P^G \models \varphi_k$ for all $G \in \mathbf{G}$.

□

For token-passing networks with specifications in $\text{ICTL} \setminus X$, Clarke et al. [2004] showed that decompositions of the form they define cannot exist in general. However, Aminof et al. [2014a] showed that by further refining the construction of Clarke et al. [2004], a non-uniform decidability result can be obtained even for specifications in $\text{ICTL}^* \setminus X$. To this end, $\text{ICTL}^* \setminus X$ is stratified not only with respect to the number k of index quantifiers, but also with respect to the number d of (nested) path quantifiers. In the following, let $k\text{-CTL}_d^* \setminus X$ be the set of all sentences in $k\text{-CTL}^* \setminus X$ with nesting depth of path quantifiers at most d .

Theorem 4.10 [Aminof et al., 2014a]. *Let \mathbf{G} be a parameterized connectivity graph. Then for all $k, d \in \mathbb{N}$, $\text{PMCP}(\mathcal{P}_{\text{simp}}^{\text{u}}, \mathbf{G}, k\text{-CTL}_d^* \setminus X)$ is decidable. Indeed, every $(\mathcal{P}_{\text{simp}}^{\text{u}}, \{\mathbf{G}\}, k\text{-CTL}_d^* \setminus X)$ has a decomposition.*

⁹Clarke et al. [2004] claimed in the discussion of their theorem that “given k and network graph G , all k -indexed LTL $\setminus X$ -specifications have the same reduction,” where a reduction is equivalent to what we call a decomposition. It should be noted that different formulas $\Phi_{\mathbb{B}}$ may be necessary for specifications with different structures of indexed quantifiers.

42 4. TOKEN-PASSING SYSTEMS

Proof idea. To show the existence of decompositions, define two graphs G, G' to be (k, d) -equivalent if they cannot be distinguished by formulas in $k\text{-CTL}_d^*\setminus X$ that use tok_i as the only atomic predicates. The existence of decompositions is proven by two observations: (i) every graph is (d, k) -equivalent to its d -contraction (a certain minimal graph with the same observed behavior), and (ii) there are only finitely many different d -contractions for a given k . By a similar argument as in the proof of Theorem 4.8, the PMCP for a given parameterized graph \mathbf{G} , process template P and specification ϕ can thus be reduced to a Boolean combination of checking whether $P^{Con} \models \phi$, for all d -contractions Con of \mathbf{G} . \square

Note that like Theorem 4.8, this result is non-uniform and non-constructive, and therefore does not supply us with an effective decision procedure for the PMCP with specifications from $\text{ICTL}^*\setminus X$. However, for specifications without quantifier alternations and for fixed \mathbf{G} (e.g., $\mathbf{G} \in \{\mathbf{R}, \mathbf{B}, \mathbf{C}\}$), the proof methods can be used to obtain concrete cutoffs for the PMCP.

We have stated before that the uniform PMCP is undecidable for specifications in $\text{ILTL}\setminus X$. For branching-time specifications, undecidability is already obtained with a fixed number of two index quantifiers, as long as the nesting depth of path quantifiers is not bounded.

Theorem 4.11 [Aminof et al., 2014a]. *There exists a parameterized connectivity graph \mathbf{G} such that $\text{PMCP}(\mathcal{P}_{simp}^u, \mathbf{G}, 2\text{-CTL}^*\setminus X)$ is undecidable.*

Proof idea. The proof is a variation of the proof of Theorem 4.9. Again, we define a parameterized graph \mathbf{G} and a sequence of formulas φ_k such that the k th TM does not halt iff φ_k holds in all system instances. The main difference is that now we can use arbitrary nestings of temporal path quantifiers to distinguish these graphs (similar to those used in Clarke et al. [2004, Theorem 5]), and therefore two index variables are sufficient to define all φ_k . \square

4.2.3 UNDECIDABILITY RESULTS FOR MULTI-VALUED TOKENS

The decidability results discussed in Sections 4.2.1 and 4.2.2 consider token-passing systems with simple token. Suzuki [1988] and Emerson and Namjoshi [1995, 2003] considered systems where the token can take multiple values.

Theorem 4.12 [Suzuki, 1988]. *Let 1-Safe be the safety fragment of $1\text{-LTL}\setminus X$. Then $\text{PMCP}(\mathcal{P}_T^u, \mathbf{R}, 1\text{-Safe})$ is undecidable for sufficiently large T .*

Proof idea. (based on Emerson and Namjoshi [2003, Section 6]) The main idea is to simulate n steps of a 2CM in a ring of size n , where one process in the ring will eventually raise a flag ‘halt’ if and only if the 2CM halts in n steps. Given a 2CM \mathcal{M} , we describe how the corresponding finite-state process P works.

In the first step, the process that starts with the token moves into a special state with the intention that it will simulate the control-state of \mathcal{M} , and it sets a flag to false. We will call this

process the *controller*. Each of the remaining processes will be used to store a fixed number of bits—these are called *storage* processes. The values in T represent different commands that the controller can send to the storage processes.

After the first step, the controller sends around the token with an “initialize” command, which makes all other processes go into storage mode. In this mode, each process stores three bit of information, corresponding to the two counters of \mathcal{M} , and a “step-counter,” respectively. Each counter is stored as a tally in the ring: for every counter, each process (except the controller) stores one bit in unary encoding, and the number of bits in the ring set to 1 is interpreted as the value of that counter. Thus, a ring of size n can store counter values up to $n - 1$. Initially, all counter-bits are set to 0. The controller implements an increment of a counter c by circulating a token with the command “increment c .” A process receiving “increment c ” will either increment its corresponding counter-bit from 0 to 1 (if possible), and pass a modified token with the “empty” command, or it will not change its internal state and pass the token unchanged (if local increment is not possible). Thus, the value of the token when it returns to the controller indicates whether the command was successful. There are similar commands for decrementing and testing for zero. After the controller implements a step of \mathcal{M} , it also increments the step-counter.

The controller proceeds in this way until its “increment step-counter” command returns to it unsuccessful (meaning that n steps of \mathcal{M} have now been simulated). If after n steps the machine \mathcal{M} did not enter a halting state, then the controller sets its flag to true. This completes the description of P . It is a finite-state process that will move into controller mode if it has the token initially, and into storage mode otherwise. The controller sets its flag to true iff \mathcal{M} does not halt in n steps. Since the non-halting problem is undecidable, so is deciding if for all n , the controller in a ring with process implementation P sets the flag to true. This proves that the PMCP is undecidable for uni-directional multi-valued token rings. \square

Binary token. Emerson and Namjoshi [2003] mentioned a simple way to simulate a multi-valued token with arbitrary set of values T using a binary token, i.e., a token with $\mathsf{T}_{\mathbb{B}} = \{0, 1\}$. Thus, they obtain another undecidability result.

Theorem 4.13 [Emerson and Namjoshi, 2003]. $\text{PMCP}(\mathcal{P}_{\mathbb{B}}^{\text{u}}, \mathbf{R}, \text{1-Safe})$ is undecidable.

Proof idea. The general proof idea is the same as in the proof of Theorem 4.12. Additionally, the controller of \mathcal{M} can use the binary token to transmit arbitrary commands from the set of commands T of that proof in a unary encoding: assume commands are numbered, i.e., we have $\mathsf{T} = [n]$ for some $n \in \mathbb{N}$. To encode command $t \in [n]$, the controller sends the token with value 1 for t times, followed by one transmission with value 0. Each storage node internally keeps track of how many 1s it received, say in a variable x . After receiving the token with value 0, every node knows which command should be executed. Now the controller again sends a 1, and every node checks if it can execute the command, i.e., whether its counter can be in- or decremented, or whether it can witness that the overall counter value is not 0. The first node that can actually

44 4. TOKEN-PASSING SYSTEMS

execute the command (or witness that the overall counter value is not 0) changes the value of the token to 0 (letting the following storage nodes know that they should not execute the command anymore). Regardless of the value of the token in this round, all nodes reset their command variable x to 0 when they pass the token. As before, the value of the token informs the controller whether the command was successful. This procedure can be repeated whenever a command is passed to the storage nodes in the original construction. \square

Multi-valued Tokens in General Graphs. Since rings are a particular form of general graphs, $\text{PMCP}(\mathcal{P}_{\text{T}\mathbb{B}}^u, \mathbf{G}, 1\text{-Safe})$ is in general undecidable. However, uni-directional rings have the characteristic property that, from the perspective of a given process, there is exactly one process from which it always receives the token, and exactly one to which it can send it. This is not the case in graphs with in- or out-degree greater than 1. Since the undecidability proof depends on the fact that the token can be used to pass certain commands to all other processes, it is not immediately obvious that it also works for other parameterized graphs. At least for bi-directional rings, we believe this is possible.

Conjecture 4.14 $\text{PMCP}(\mathcal{P}_{\text{T}\mathbb{B}}^u, \mathbf{B}, k\text{-LTL}\setminus X)$ is undecidable.

For general graphs, the problem is mostly open.

Problem 4.15 For which parameterized connectivity graphs \mathbf{G} is $\text{PMCP}(\mathcal{P}_{\text{T}\mathbb{B}}, \mathbf{G}, \mathcal{F})$ decidable, for $\mathcal{F} \in \{\text{ILTL}\setminus X, \text{ICTL}^*\setminus X\}$?

4.3 RESULTS FOR DIRECTION-AWARE TOKEN-PASSING SYSTEMS

In the results considered thus far, if a node has multiple recipients, then it cannot choose which one will receive the token, but one of the possible transitions will fire non-deterministically. In this section, we consider systems that are direction-aware, i.e., where processes can choose who will receive the token and/or from whom they want to receive it. We first give a decidability result for a special class of systems that can be used to model a leader election protocol [Emerson and Kahlon, 2004], and then a general undecidability result.

4.3.1 CUTOFFS FOR CHANGE-BOUNDED TOKENS IN BI-DIRECTIONAL RINGS

Let \mathbf{B} be the parameterized connectivity-graph of bi-directional rings of size n , i.e., $\mathbf{B}(n) = (V, E)$ with $V = [n]$ and $E = \{(i, j), (j, i) : j = i + 1 \bmod n\}$. Let $Dir_{out} = \{\text{cw}, \text{ccw}\}$, and dir_{out} the function that maps edges $(i, i + 1 \bmod n)$ to cw, and edges $(i + 1 \bmod n, i)$ to ccw. Let $Dir_{in} = \{\text{undefined}\}$, and dir_{in} be the function that maps all edges to undefined. Let \mathbf{B}^{rnd} be the parameterized bi-directional ring with these direction labels. Let $\mathcal{D}_{\text{T}}^{\text{rnd}}$ be the set of all deterministic process templates in \mathcal{P}_{T} .

We call a parameterized (direction-aware) TPS $(\mathcal{P}_T, \{G\})$ *change-bounded* if there exists $b \in \mathbb{N}$ such that, for every run of every system instance $\mathcal{P}_T^{G(n)}$, the value of the token does not change more than b times.¹⁰

Theorem 4.16 [Emerson and Kahlon, 2004]. $\text{PMCP}(\mathcal{D}_T^{\text{snd}}, \mathbf{B}^{\text{snd}}, \text{1-LTL} \setminus X)$ is decidable if $(\mathcal{D}_T^{\text{snd}}, \mathbf{B}^{\text{snd}})$ is change-bounded. Indeed, the cutoff is a polynomial depending on the number of states and the transition graph of process template P .

Proof idea. The proof is based on the fact that processes are deterministic, and so the token can only make a finite number of moves until the system either deadlocks or runs into a state that it has seen before, entering an infinite loop. The number of steps until the system reaches such a state is independent of the size of the ring, as long as it has a certain minimal size. This size can be computed by constructing the transition graph for the ring, based on the definition of P . It gives a cutoff for $\text{PMCP}(\mathcal{D}_T^{\text{snd}}, \mathbf{B}^{\text{snd}}, \text{1-LTL} \setminus X)$. \square

Note that this result is not displayed in Figure 4.3, since it considers special restrictions to deterministic process templates and change-bounded tokens.

4.3.2 UNDECIDABILITY FOR DIRECTION-AWARE TPSS

Since direction-aware TPSs can degenerate to direction-unaware TPSs, the undecidability results from Section 4.2.3 also hold here, and decidability can only be obtained by adding restrictions.

Note that while Theorem 4.16 gives a cutoff for a system with limited direction-awareness and multi-valued tokens, it is very restricted in the specifications, the processes and the graphs (including direction-awareness) that it can handle. If we consider $|Dir_{in}| > 1$, Aminof et al. [2014a] showed that undecidability follows even for systems with a simple token.

Now, consider the case $Dir_{out} = Dir_{in} = \{cw, ccw\}$, and $dir_{out} = dir_{in}$ the function that maps edges in clockwise directions to cw, and edges in counterclockwise direction to ccw. Let \mathbf{B}^{dir} be the parameterized bi-directional ring with these direction labels.

Theorem 4.17 [Aminof et al., 2014a]. $\text{PMCP}(\mathcal{P}_{simp}, \mathbf{B}^{dir}, \text{1-LTL} \setminus X)$ is undecidable.

Proof idea. The proof goes along the lines of the proofs of Theorems 4.12 and 4.13. The main new idea is that directions can be used to encode the different commands of the controller to the storage nodes, similar to the encoding with a binary token. \square

¹⁰An example is an integer-valued token with fixed starting value b , that can only be decreased, and never become smaller than 0.

46 4. TOKEN-PASSING SYSTEMS

Furthermore, undecidability with respect to $k\text{-LTL}\backslash X$ has been shown even if only one of $|Dir_{in}| > 1$ or $|Dir_{out}| > 1$ holds, for sufficiently large k .

Theorem 4.18 [Aminof et al., 2014a]. *Assume that $|Dir_{out}| > 1$. Then there exist a parameterized directed connectivity graph \mathbf{G} such that $\text{PMCP}(\mathcal{P}_{simp}^{\text{snd}}, \mathbf{G}, 9\text{-LTL}\backslash X)$ is undecidable. Similarly, for $|Dir_{in}| > 1$ and suitable \mathbf{G}' , $\text{PMCP}(\mathcal{P}_{simp}^{\text{rcv}}, \mathbf{G}', 9\text{-LTL}\backslash X)$ is undecidable.*

Proof idea. The proof is again a reduction from the non-halting problem of 2CMs and requires a parameterized graph with eight special nodes that are directly connected to the controller, and represent the different commands of the 2CM (increment counter, decrement counter, counter is zero, counter is not zero; each for counters 1 and 2). The other nodes are storage nodes, like in the undecidability proofs before.

Consider the case where $|Dir_{out}| = 1$: since all edges have the same outgoing label, processes cannot choose where to send the token. We use a special construction of the parameterized graph and the process template such that every possible path of the token through the graph corresponds to the (partial) execution of one command, represented by the special node that the token visits after leaving the controller. To circumvent the restriction that the controller cannot choose the sending direction, we use a specification that tracks whether the special node that the token is sent to is always the one that the controller intended (represented by the state of the controller). This ensures that we only consider runs of the system that correspond to an execution of the 2CM.

For $|Dir_{in}| = 1$ we can use a similar construction, except that now the controller can issue a certain command, but the storage processes cannot detect which one is intended, and have to guess which command to execute. The special nodes are in this case visited immediately before the token returns to the controller. Again, we use the specification to ensure that only the intended runs are considered. \square

4.4 DISCUSSION

Even in systems with a relatively restricted communication primitive such as passing a valueless token, the PMCP is undecidable if we do not additionally restrict the graph structure or stratify specifications with respect to the number of index and path quantifiers. Undecidability proofs show that such systems can simulate a 2CM, and thus halting of the 2CM could be decided if we could decide the PMCP.

For certain concrete classes of graphs, such as rings, cutoffs can be found by hand. To prove that c is a cutoff for a class of systems and specifications in $\text{ICTL}^*\backslash X$, it suffices to show that there is a stuttering bisimulation between a system with arbitrarily many components and a system with exactly c components. For specifications that quantify over multiple processes, this needs to take into account whether processes are direct neighbors or not, and consider all possible cases.¹¹

¹¹In fact, Aminof et al. [2014b] recently showed that cutoffs can be computed automatically for large classes of graphs that are definable either by graph-grammars or in Monadic Second-Order Logic. Examples include rings, cliques, lines, trees,

If tokens can carry values, decidability is lost even in rings. Similarly, if processes can distinguish directions in the graph, we get undecidability in bi-directional rings. Figure 4.3 (on page 37) and Table 4.1 give an overview of decidability results for token-passing systems with these properties, showing that most of the considered cases are undecidable. To recover decidability, additional restrictions on graphs, processes, or specifications, are necessary.

Table 4.1: Decidability results for TPSs and their sources, separated into systems with direction-unaware processes and simple token (\mathcal{P}_{simp}^u), systems with direction-unaware processes and multi-valued token ($\mathcal{P}_T^u, \mathcal{P}_{TB}^u$), and systems with direction-aware processes

Result	Processes	Graph	Specification	References
a cutoff	\mathcal{P}_{simp}^u	\mathbf{R}	fragments of $\text{ICTL}^* \setminus X$	[Emerson and Namjoshi 2003] [Aminof et al. 2014a]
decidability for all k	\mathcal{P}_{simp}^u	$\forall \mathbf{G}$	$k\text{-LTL} \setminus X$	[Clarke et al. 2004]
decidability for all k, d	\mathcal{P}_{simp}^u	$\forall \mathbf{G}$	$k\text{-CTL}_d^* \setminus X$	[Aminof et al. 2014a]
undecidability	\mathcal{P}_{simp}^u	$\exists \mathbf{G}$	$\text{ILTL} \setminus X,$ $2\text{-CTL}^* \setminus X$	[Aminof et al. 2014a]
undecidability	\mathcal{P}_T^u	\mathbf{R}	1-Safe	[Suzuki 1988]
undecidability	\mathcal{P}_{TB}^u	\mathbf{R}	1-Safe	[Emerson and Namjoshi 2003]
decidability under bounded change	\mathcal{D}_T^{snd}	\mathbf{B}^{snd}	$1\text{-LTL} \setminus X$	[Emerson and Kahlon 2004]
undecidability	\mathcal{P}_{simp}	\mathbf{B}^{dir}	$1\text{-LTL} \setminus X$	[Aminof et al. 2014a]
undecidability for $ Dir_{out} > 1$, for $ Dir_{in} > 1$	$\mathcal{P}_{simp}^{snd},$ \mathcal{P}_{simp}^{rcv}	$\exists \mathbf{G}$	$9\text{-LTL} \setminus X$	[Aminof et al. 2014a]

For all of the decidability results in this chapter, a restriction to some notion of fairness is essential. While some of the original results are explicitly restricted to systems or runs where every process receives and sends the token infinitely often, we argue here (and in Aminof et al. [2014a]) that a weaker condition is sufficient, namely that from every state of the process there *exists* a path such that it sends or receives the token—but results are not restricted to runs where such paths

series-parallel graphs, but not grids. From a description (i.e., a grammar or a formula) of such a set of graphs, one can build an algorithm that computes a cutoff for a given formula and thus solves the PMCP for indexed $\text{CTL}^* \setminus X$ over this set of graphs.

48 4. TOKEN-PASSING SYSTEMS

are actually taken. There are no decidability results in the literature that consider systems without any fairness assumption.

4.4.1 VARIATIONS OF THE MODEL

The basic idea behind structuring concurrent applications using a token is that the current holder of the token is privileged, and thus allowed to perform some special action such as entering its critical section. That is, tokens are a means to resolve conflicts over shared resources. However, systems with a single token are based on the assumption that at each time at most one process may be privileged, or equivalently, there is only one shared resource. This severely limits the degree of concurrency in the system.

There are several schemes to increase the degree of concurrency. For instance, any solution to the celebrated dining philosophers problem allows several (non-neighboring) philosophers to eat at the same time. A generalization of dining philosophers are the drinking philosophers [[Chandy and Misra, 1984](#)] for general conflict graphs, where again one requires that two neighbors are not drinking at the same time, in case both are currently competing for the same resource. In the extreme case, a completely connected conflict graph then boils down to having at most a single neighbor drinking at a time, and thus to a single token scheme. Intuitively, the dining (or drinking) philosophers on general graphs can thus be interpreted as a token scheme with multiple tokens.

The literature contains results on both areas, access to multiple shared resources and multiple tokens. Regarding multiple shared resources, there are several cutoff results. Ensuring fair access is considered by [Emerson and Kahlon \[2002\]](#), FIFO access is considered by [Bouajjani et al. \[2008\]](#), and strict alternation rules as provided by link reversal algorithms [[Barbosa and Gafni, 1989](#)] is considered by [Függer and Widder \[2012\]](#). Regarding multiple tokens, [Emerson and Kahlon \[2004\]](#) contains a decomposition result that allows to reduce reasoning about 2-indexed properties of uni-directional rings with multiple multi-valued tokens, to reasoning in small rings. To obtain decidability, there are several restrictions on token-passing and the processes: (i) in any run, every multi-valued token can only change its value a bounded number of times, (ii) every token has an “owner” process, and transitions that send or receive a token cannot change the state of a process unless the token is owned by it, and (iii) processes have to be deterministic.

Considering the case with multiple tokens, we conjecture that decidability strongly depends on whether or not processes can distinguish different tokens. If this is the case, then the problem is very similar to a multi-valued token and should quickly become undecidable. For indistinguishable tokens, there is more hope to obtain decidability results.

Problem 4.19 Do any of the decidability results in this chapter still hold for systems with multiple tokens? In particular, do they hold if processes *cannot* distinguish different tokens?

Furthermore, all previous work on TPSs only considers prenex ITL. An interesting question is under which circumstances the given results hold if we allow index quantifiers inside of temporal quantifiers.

Problem 4.20 Are there fragments of non-prenex ITL that can express interesting properties that are not in $\text{ICTL}^* \setminus X$ and still have a decidable PMCP for any of the cases considered in this chapter?

CHAPTER 5

Rendezvous and Broadcast

This chapter considers systems of processes communicating via pairwise rendezvous, asynchronous rendezvous, broadcast, and combinations of these (as defined in Section 2.2.1). We have already seen in Examples 2.1 and 2.2 how pairwise-rendezvous models standard synchronization between exactly two processes. As discussed by Delzanno et al. [2002], asynchronous rendezvous and broadcast also correspond to standard coordination constructs in different programming languages, such as the `notify` and `notifyAll` constructs of the Java programming language.

The specifications considered in this chapter are either action-based, i.e., $\omega\text{Reg}(A)$ and $\text{Reg}(A)$, or state-based, i.e., $\text{LTL}(C)$ which are LTL formulas whose atoms are indexed by the controller process, and $\forall U \text{ LTL}(U)$ which are LTL formulas that hold for every user process.

Figure 5.1 and Table 5.1 (on page 53) summarize results taken from the literature [Emerson and Kahlon, 2003b, Esparza et al., 1999, German and Sistla, 1992], including minor extensions. The figure indicates that safety is always decidable, while liveness is decidable for pairwise rendezvous and undecidable for the other communication primitives.

Running example. As the running example of this chapter, we will revisit the simple Client/Server system from Chapter 2, including a number of variants with different communication primitives.

5.1 SYSTEM MODEL

Synchronization constraints. As in Section 2.2.1, pairwise rendezvous is defined by taking $\text{card} = \{1\}$, asynchronous rendezvous is defined by taking $\text{card} = \{0, 1\}$, and broadcast is defined by taking $\text{card} = \mathbb{N}_0$. Systems with all three primitives, for example, are defined by taking $\overline{\text{card}} = \{\{1\}, \{0, 1\}, \mathbb{N}_0\}$.

Example 5.1 Client/Server (revisited) Recall the simple client/server system from Examples 2.1 and 2.2. Example 2.2 illustrates pairwise rendezvous where the server can synchronize with a single client at a time.

For asynchronous rendezvous, there are two possibilities if a client is ready to take a transition labeled with $\text{out}_{\text{enter}}$ or $\text{out}_{\text{leave}}$. First, the server and the client take a synchronized transition, as in pairwise rendezvous. The second case may occur when the client is ready to effectuate a transition labeled with $\text{out}_{\text{enter}}$ or $\text{out}_{\text{leave}}$, but from the current state of the server there is no outgoing

52 5. RENDEZVOUS AND BROADCAST

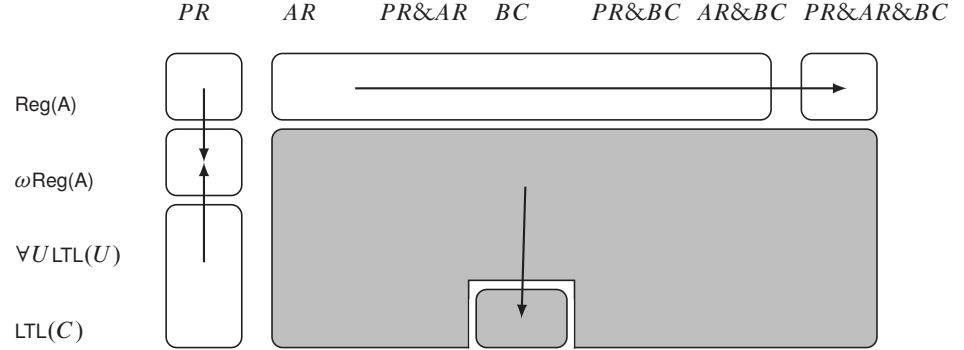


Figure 5.1: PMCP results for systems with a controller, and clique connectivity-graph. The abbreviations PR, AR, BC stand for pairwise rendezvous, asynchronous rendezvous, broadcasts. White boxes are decidability results, dark boxes are undecidability results. An arrow from box A to box B means that the result of A follows from the result, or variation of the proof, of B.

transition labeled with $i n_{\text{enter}}$ or $i n_{\text{leave}}$, respectively. In this case, the client may effectuate its transition alone, that is, without the server. One may view this as the client transmitting information to the server in case the server is ready to receive one, otherwise the message is “lost.”

For broadcast, consider the connectivity graph in Figure 5.2(a), and the process templates in Figures 5.2(b) and 5.2(c), where the server has a transition from 0_S labeled with $\text{enter}!!$, and all the clients have transitions from 0_C labeled with $\text{enter}??$. In this case, the synchronization is initiated by the server, and any client that is ready to take the synchronous step can enter 1_C .

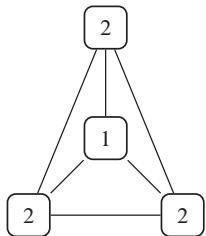
Parameterized connectivity graph. A graph $G = (V, E)$ is a *clique* if for all distinct $v, w \in V$ it holds that $(v, w) \in E$. The 2-ary *parameterized clique graph* is the sequence $n \mapsto \mathbf{C}_\circ(n)$ where $\mathbf{C}_\circ(n)$ is the clique with $n + 1$ vertices, type(1) = 1, and type(i) = 2 for all $i \neq 1$. The unique process in $\mathbf{C}_\circ(n)$ of type 1 (it has index 1) is called the *controller*, and all other processes are called *users* and are of type 2.

Example 5.2 Clique topology for Client/Server system Figure 5.2(a) shows a clique topology with one controller process (modeling the server), and three user processes (modeling the clients). Note that with the process templates defined for server and clients in Figures 2.1(b) and 2.1(c), the behavior of the composed system does not change, since the process templates do not define synchronous transitions between two clients.

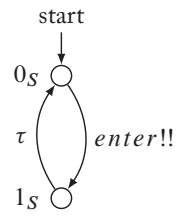
Runs and Deadlocks. In this chapter, system runs are defined as *infinite* paths that start in the initial state (cf. Section 2.2.2). The only exception is Theorem 5.7 about the action-based specification language $\text{Reg}(A)$ which is interpreted over finite paths that start in the initial state.

Table 5.1: Decidability results, and their sources, for systems with a controller, using pairwise rendezvous (PR), asynchronous rendezvous (AR), and broadcast (BC)

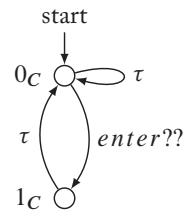
Result	Processes	Graph Specification	References
decidability	PR+BC+AR	C_{\odot}	$\text{Reg}(A)$ [Esparza et al. 1999] [Emerson and Kahlon 2003b]
decidability in EXPSPACE	PR	C_{\odot}	$\omega\text{Reg}(C)$ [German and Sistla 1992]
decidability in EXPSPACE	PR	C_{\odot}	$\forall U \omega\text{Reg}(U)$ [German and Sistla 1992]
decidability in EXPSPACE	PR	C_{\odot}	$\omega\text{Reg}(A)$ [Emerson and Kahlon 2003b] cf. [German and Sistla 1992]
undecidability	BC	C_{\odot}	$\omega\text{Reg}(A)$ [Esparza et al. 1999]
undecidability	BC	C_{\odot}	$\text{LTL}(C)$ [Emerson and Kahlon 2003b]
undecidability	BC	C_{\odot}	$\forall U \text{LTL}(U)$ cf. [Esparza et al. 1999]
undecidability	AR	C_{\odot}	$\omega\text{Reg}(A)$ [Emerson and Kahlon 2003b]
undecidability	AR	C_{\odot}	$\text{LTL}(C)$ [Emerson and Kahlon 2003b]
undecidability	AR	C_{\odot}	$\forall U \text{LTL}(U)$ cf. [Emerson and Kahlon 2003b]



(a) 2-ary clique graph with 3 user vertices and one controller vertex.



(b) Modified process template S' for the server.



(c) Modified process template C for the clients.

Figure 5.2: Clique graph and modified process templates for client/server system.

German and Sistla [1992, Theorem 3.19] proved that deadlock detection is decidable for systems using pairwise rendezvous, by a reduction to the reachability problem for VASS. To the

54 5. RENDEZVOUS AND BROADCAST

best of our knowledge, this is the only result on parameterized deadlock detection for the systems in this chapter.

Specifications. Write $\forall U \text{LTL}(U)$ for the set of all LTL sentences of the form $\forall i : \text{type}(i) = 2. \phi(i)$. The interpretation of such a formula in $\mathbf{C}_\odot(n)$ is that for every process index i of type 2 (i.e., for every “user” process) the projection of every infinite path π onto coordinate i satisfies $\phi(i)$.

Write $\text{LTL}(C)$ for the set of all LTL formulas of the form $\phi(1)$. The interpretation of such a formula in $\mathbf{C}_\odot(n)$ is that the projection of every infinite path π onto the unique controller process satisfies $\phi(1)$.

More generally, denote by $\omega\text{Reg}(C)$ the specification languages of the form $AL(1)$ where L is an ω -regular language over $\text{AP}_{\text{pr}} \times \{1\}$ (recall the controller has index 1), and denote by $\forall U \omega\text{Reg}(U)$ the specification languages of the form $\forall i : \text{type}(i) = 2. AL(i)$ where L is an ω -regular language over alphabet $\text{AP}_{\text{pr}} \times \{i\}$ (recall the users have type 2).

The set of action-based specifications, i.e., $\text{Reg}(A)$ and $\omega\text{Reg}(A)$, are defined in Section 2.4.2.

Example 5.3 Specifications for Client/Server system. An interesting safety property for the clients of the system is mutual exclusion, i.e., no two clients should be in state 1_C at the same time. Note that none of the specification types above allows one to directly express this property. However, as noted by German and Sistla [1992], we can indirectly check for it by modifying the process template: add a state 2_C to the client template that can only be entered from 1_C by synchronizing on a new action fail with another client process that is also in 1_C . In this way, a process can enter 2_C iff we reach a state where at least two processes are in 1_C at the same time. Then, mutual exclusion is expressed by the formula

$$\forall U \mathbf{G}\neg 2_C(U),$$

where we use states of a process template as atomic propositions. Alternatively, we can express violation of the property as the regular expression over actions

$$\{a, \text{enter}, \text{leave}\}^* \text{ fail.}$$

In addition, we can specify liveness properties, e.g., every client should enter 1_C infinitely often. This is expressed by the formula

$$\forall U \mathbf{GF}1_C(U).$$

In the following two sections, we will see that the mutual exclusion property can be decided for systems with any combination of pairwise rendezvous, asynchronous rendezvous, and broadcast. In contrast, liveness properties like the one above can in general only be decided in systems that are restricted to pairwise rendezvous.

5.2 DECIDABILITY RESULTS

All the decidability results in this chapter follow the same idea of building a transition system which is a counter representation of the parameterized system and the parameterized specification. In Proposition 5.4 we show that for communication primitives including the ones dealt with in this chapter (namely broadcast, pairwise rendezvous, asynchronous rendezvous) this transition system is a well-structured transition system. The main point to be checked is monotonicity. Moreover, in the case of pairwise rendezvous, this transition system is a vector addition system with states (VASS).

5.2.1 COUNTER REPRESENTATION

The idea of using a counter representation already appears in the work of German and Sistla [1992], and relies quite heavily on the connectivity graph being a clique.

Suppose the process template of the controller has state set Q_C , the process template of the user processes has state set Q_U , and the specification is given by an (finite-state or Büchi) automaton A over the atomic propositions of the controller, and with state set Q_A . Recall from Vardi [1995] that for every LTL formula φ over atomic propositions AP there is a non-deterministic Büchi automaton over alphabet 2^{AP} , of size $2^{O(|\varphi|)}$, that accepts exactly the infinite strings satisfied by φ .

Informally, the counter representation captures the current state of the controller (an element of Q_C), the current state of the automaton (an element of Q_A), and the number of user processes in each state of Q_U (a vector of non-negative integers of dimension $|Q_U|$). Thus, the states of the counter representation are of the form (q_c, q_a, \bar{v}) , and transitions between these states mimic global transitions in system instances.

Definition of the Counter Representation M . Suppose $Q_U = \{t_1, \dots, t_k\}$, and automaton A has initial state $\iota_A \in Q_A$, and accepting state set $F \subseteq Q_A$. Also, suppose $s \in Q_C \times (Q_U)^n$ is a global state of a system instance with n user processes, and q_A is a state of the automaton A . Define the *abstracted state* $\text{abs}(s, q_A)$ as the tuple $(q_C, q_A, x_1, \dots, x_{|Q_U|})$ in $Q_C \times Q_A \times \mathbb{N}_0^{|Q_U|}$ where q_C is $s(1)$ (the local state of the controller process), and for $1 \leq i \leq |Q_U|$, $x_i := \#\{j : s(j) = t_i\}$, i.e., the number of user processes that are in local state t_i in the global state s .

The *counter representation* is a transition system $M = (Q, \Sigma, \Delta)$ defined as follows.

- The *state set* Q of M is defined as $Q_C \times Q_A \times \mathbb{N}_0^{|Q_U|}$, i.e., the set of all abstracted states $\text{abs}(s, q_A)$.
- The set of *action labels* Σ of M is the set $\Sigma_{int} \cup \Sigma_{sync}$, i.e., the action labels of the original system.
- The *transitions* Δ of M are of the form $\text{abs}(s, q_A) \xrightarrow{a} \text{abs}(s', q'_A)$ with $a \in \Sigma$, where
 - (i) $s \xrightarrow{a} s'$ is a global transition of the original system, and

56 5. RENDEZVOUS AND BROADCAST

- (ii) if this transition does not involve the controller then $q'_A = q_A$, else q'_A can be reached from q_A in one step on input $\{p \in AP_{\text{pr}} : p \text{ holds in state } q_C\}$. Thus the state q_A evolves according to the label of the state q_C .

If q_A is in the accepting set F then we say that the abstracted transition *hits* F .

- The set of *initial states* consists of all states of the form $(\iota_C, \iota_A, \bar{v})$ where ι_C is an initial state of the controller template, ι_A is the initial state of automaton A , and \bar{v} is an arbitrary vector whose support S (i.e., the set of coordinates with non-zero entries) is non-empty and $s \in S$ iff t_s is an initial state of the user process template. Note that the set of initial states is computable and downward closed, and thus we satisfy the hypothesis of Theorem 3.3(2).

Note that the transition system M , although infinite, is computable, meaning that one can compute the state set and the transition relation.

Properties of the Counter Representation M . We elaborate on the properties of the counter representation that will later be used in decidability proofs.

Proposition 5.4 If card is an interval then M is a WSTS. Let \leq be the ordering on the states of M defined by $(q_C, q_A, \bar{v}) \leq (q'_C, q'_A, \bar{v}')$ if $q_C = q'_C$, $q_A = q'_A$ and for all i , $v_i \leq v'_i$. If card is an interval, namely of the form $[a, b]$ or $[b, \infty)$, then (i) the ordering \leq is monotone with respect to the transition system M and (ii) one can compute, from a basis for an upward closed set C , a basis for $\text{Pred}(C)$.

Proof Sketch. The first item holds under very general computability conditions, namely if card is computable and the global transition relation Δ is computable (see computability assumption 2.5.1). Indeed: if B is a basis for C , then a basis $\text{Pred}(C)$ consists of all configurations \bar{v} such that there is some action label a and a global transition from \bar{v} to some element of B on action a .

For the second item we fix some notation. Define an operation $+$ on states of M that have the same first two coordinates, namely, $(q, r, \bar{v}) + (q, r, \bar{w}) := (q, r, \bar{v} + \bar{w})$ where addition is componentwise. From now on we drop all mention of the first two coordinates. The states of U are $\{t_1, \dots, t_k\}$. For $i \leq k$, write e_i for the vector \bar{v} such that $v_i = 1$ and $v_j = 0$ for all $j \neq i$.

For the second item, suppose $\bar{v} \xrightarrow{a} \bar{w}$ is a transition of M induced by a local transition $t_i \xrightarrow{a_{out}} t_j$. Then we may write $\bar{v} = e_i + \bar{c}$ and $\bar{w} = e_j + \bar{d}$ for some vectors \bar{c}, \bar{d} . Suppose that exactly N processes synchronize with the initiating process in this transition. Then $N \in \text{card}$ and either $N = \max(\text{card})$; or $N < \max(\text{card})$ and $N + 1 \in \text{card}$ (because card is an interval) and, by condition (MAX), there are no more processes in the state represented by \bar{v} that can synchronize on an a -action (\dagger).

Now, to establish monotonicity, suppose $\bar{v} \leq \bar{y}$. We should prove that there is an a -transition from \bar{y} to some $\bar{z} \geq \bar{w}$. It is enough to consider the case that $\bar{y} = \bar{v} + e_l$ for some l .

If $N = \text{max}(\text{card})$ then we may define $\bar{z} := \bar{w} + e_l$ (i.e., mimic the transition $\bar{v} \xrightarrow{a} \bar{w}$, and the new process in state t_l does not synchronize).

Now assume $N < \text{max}(\text{card})$. There are two cases.

First case: action a is not enabled from state t_l , i.e., the local state t_l has no outgoing transition labeled a_{in} . In this case we may define $\bar{z} := \bar{w} + e_l$ (i.e., mimic the transition $\bar{v} \xrightarrow{a} \bar{w}$, and the new process in state t_l does not, in fact cannot, synchronize)

Second case: action a is enabled from state t_l , say $t_l \xrightarrow{a_{in}} t_m$ is a local transition. In this case we may define $\bar{z} := \bar{w} + e_m$. Indeed, in the transition from \bar{y} to \bar{z} exactly $N + 1 \in \text{card}$ processes synchronize with the initiating process, and if it were possible that more than $N + 1$ were available to synchronize, then more than N would have been available to synchronize in the transition from \bar{v} , contrary to (\dagger) . \square

In other words, the combined system in which card is an interval and the specification is given by an automaton is a WSTS. Note that card is an interval for pairwise rendezvous ($\{1\}$), asynchronous rendezvous ($\{0, 1\}$), and broadcast (\mathbb{N}_0). The next lemma allows one to combine WSTS from different communication primitives into a single WSTS.

Lemma 5.5 *If $M_i = (Q, \Sigma_i, \Delta_i)$ ($i = 1, 2$) are two transition systems over the same state set Q , and with disjoint sets of action labels Σ_i , and \leq is an ordering on Q , and both M_1 and M_2 are WSTS with respect to \leq , then the transition system $M_1 \cup M_2 := (Q, \Sigma_1 \cup \Sigma_2, \Delta_1 \cup \Delta_2)$ is a WSTS with respect to \leq .*

Sketch. First, $M := M_1 \cup M_2$ is computable since the M_i are. Second, \leq is monotonic with respect to M since the sets of action labels of the M_i s are disjoint. Third, $\text{Pred}_M(C) = \text{Pred}_{M_1}(C) \cup \text{Pred}_{M_2}(C)$. From a basis for C compute a basis B_i for $\text{Pred}_{M_i}(C)$. Then $B_1 \cup B_2$, which is certainly computable, is a basis for $\text{Pred}_M(C)$. \square

Corollary 5.6 *Let M be the counter representation of the parameterized system $n \mapsto \text{sys}(\overline{P}, \mathbf{G}(n), \text{card})$ where \overline{P} consists of the set of all 2-ary system templates, $\text{card} = \{\{1\}, \{0, 1\}, \mathbb{N}_0\}$ and \mathbf{G} is the 2-ary clique graph of size n . Then M is a WSTS.*

Aside. If card is not an interval then M may not be a WSTS under \leq . This is due to the (MAX) item in the definition of synchronous transition (p. 8). For instance, suppose $\text{card} = \{1, 3\}$ and the process template has transitions $q \xrightarrow{a_{out}} q$ and $s \xrightarrow{a_{in}} s'$. Let c be the configuration in which one process is in state q and two processes are in state s . Then there is a synchronous transition to configuration c' in which one process is in state q , one process is in state s , and one process is in state q' . Now, consider configuration d in which one process is in state q and three processes are in state s . Then $c \leq d$. However the only synchronous transition from d results in a configuration d' in which one process is in state q , and none in state s — this is because the (MAX) item forces all

58 5. RENDEZVOUS AND BROADCAST

three processes in state s to receive the broadcast. But it is not the case that $c' \leq d'$, and so \leq is not monotone.

On the other hand, if synchronous transitions are defined without the (MAX) item then M is a WSTS for every card (since in the proof of Proposition 5.4 we may simply take $\bar{z} = \bar{w} + e_l$).

5.2.2 DECIDABILITY FOR ALL THREE PRIMITIVES

We justify the decidability entry in the PR&AR&BC column of Figure 5.1.

Theorem 5.7 [Esparza et al., 1999, Section 4] [Emerson and Kahlon, 2003b, Section 4.1] *If synchronization is by pairwise rendezvous, asynchronous rendezvous, and broadcast, then $\text{PMCP}(\mathcal{P}, \mathbf{C}_\odot, \text{Reg}(A))$ is decidable.*

Proof. Let A be an automaton for the complement of the specification ϕ , and build the WSTS M as above. By Corollary 5.6, M is a WSTS. Let C be the finite set $Q_P \times F_A \times (0, \dots, 0)$, where F_A are the final states of the automaton A . There is a path from $(\iota_P, \iota_A, 0, \dots, 0)$ to $\uparrow C$ iff the specification ϕ fails on some system instance of the parameterized system. Now use the fact that coverability is decidable for WSTS (Theorem 3.3). \square

5.2.3 DECIDABILITY FOR PAIRWISE RENDEZVOUS

We discuss the decidability results in the PR column of Figure 5.1.

Theorem 5.8 [German and Sistla, 1992, Section 3.2] *If synchronization is by pairwise rendezvous then $\text{PMCP}(\mathcal{P}, \mathbf{C}_\odot, \omega\text{Reg}(C))$ and $\text{PMCP}(\mathcal{P}, \mathbf{C}_\odot, \text{LTL}(C))$ are decidable in EXPSPACE.*

Sketch. Let A be a Büchi automaton over the atomic propositions of the controller. We build the WSTS M as above, except that we introduce a unique initial state $init$, with transitions to all states of the form $(\iota_C, \iota_A, (0, \dots, 0))$ where ι_C is an initial state of the controller and ι_A is the initial state of the automaton, as well as transitions from $(\iota_C, \iota_A, \bar{v})$ to $(\iota_C, \iota_A, \bar{w})$ where \bar{w} is equal to \bar{v} except that at some coordinate, say i , $w_i = v_i + 1$ and t_i is an initial state of the user process, and from each of the states $(\iota_C, \iota_A, \bar{v})$ there are transitions which begin the simulation of the original system. Informally, M first loads the size of the system and then simulates it. In addition, one has to make sure that global transitions with no effect on the counters are correctly simulated. This is done by splitting each transition into two: first the coordinates of the source-states are decremented by 1 and then the coordinates of the target-states are incremented by 1.

By Corollary 5.6, M is a WSTS. It is not hard to show that M is the configuration space of a vector addition system with states (VASS). The reason for this is that transitions of the original system (internal transitions and pairwise-rendezvous) translate to adding constant vectors to states of M . Actually, one has to make sure that global rendezvous transitions with no net-effect on the counters do not result in extra behaviors. This is done by splitting each VASS transition into two: first the co-ordinates of the source-states are decremented by 1 (recall that a decrement transition

can only be taken in a VASS if the co-ordinates being decremented are non-zero) and then the co-ordinates of the target-states are incremented by 1.

The problem “is there an $n \in \mathbb{N}$ and an infinite run in the system with n users that is accepted by automaton A ” is equivalent to the problem of whether some path of the VASS M hits F infinitely often. This is a control state repeated reachability problem, which is decidable in EXPSPACE for VASS (Theorem 3.4).

Note that if the specification is given as an LTL formula ϕ over the controller then the first step is to translate the formula into a non-deterministic Büchi automaton L of size $2^{O(|\phi|)}$. In this case the VASS is $O(|U|)$ -dimensional and has state set of size $|Q_C| \times 2^{O(|\phi|)}$. Thus, by Theorem 3.4 we can solve PMCP in space polynomial in $|Q_C| \times 2^{O(|\phi|)}$ and exponential in $|U|$, thus in EXPSPACE. \square

By simulating a user process in the controller and using the fact that each user has the same set of neighbors, one gets the following.

Theorem 5.9 [German and Sistla, 1992, Section 3.2] *If synchronization is by pairwise rendezvous then $\text{PMCP}(\mathcal{P}, \mathbf{C}_\odot, \forall U \omega\text{Reg}(U))$ is decidable in EXPSPACE. In particular, the result holds for $\forall U \text{LTL}(U)$.*

A proof of the following theorem is hinted at by Emerson and Kahlon [2003b]. It can be proven in a similar way to Theorem 5.8.

Theorem 5.10 [Emerson and Kahlon, 2003b, Section 5.2] *If synchronization is by pairwise rendezvous then $\text{PMCP}(\mathcal{P}, \mathbf{C}_\odot, \omega\text{Reg}(A))$ is decidable in EXPSPACE.*

5.3 UNDECIDABILITY RESULTS

We discuss the undecidability results in the BC and AR columns of Figure 5.1. The rest of the undecidability results are then immediate. The basic idea is to reduce the non-halting problem of 2CMs to the PMCP by simulating the states of the 2CM in the controller and distributing the counters over the users.

5.3.1 UNDECIDABILITY FOR BROADCAST

We discuss the undecidability results in the BC column of Figure 5.1. The proofs of the following results are all similar, and follow Esparza et al. [1999, Section 5].

Theorem 5.11 *Suppose synchronization is by broadcast. Then:*

1. $\text{PMCP}(\mathcal{P}, \mathbf{C}_\odot, \text{LTL}(C))$ is undecidable;
2. $\text{PMCP}(\mathcal{P}, \mathbf{C}_\odot, \omega\text{Reg}(A))$ is undecidable; and
3. $\text{PMCP}(\mathcal{P}, \mathbf{C}_\odot, \forall U \text{LTL}(U))$ is undecidable.

60 5. RENDEZVOUS AND BROADCAST

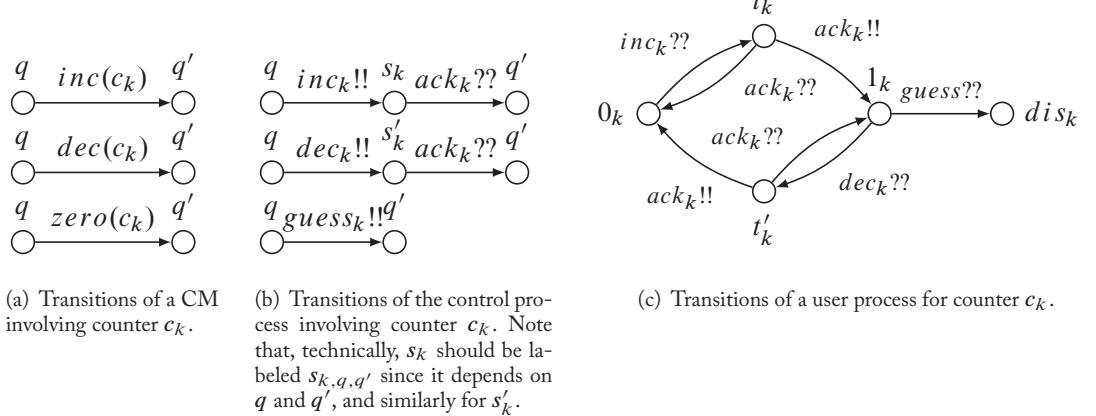


Figure 5.3: Simulating a CM using broadcasts.

Proof idea. We explain the first item. The second is similar, and we remark at the end how to deal with the third item. Suppose M is a 2CM with a halting location m , and without loss of generality the starting location 1 cannot be reached by a transition from any location. We build a new 2CM M' that simulates M but if the halting location m is reached then both counters are drained to zero and then there is a transition to the initial location. Thus, M' never stops and has the following property: M has a computation that enters location m if and only if M' has a computation that enters the location m infinitely often. The protocol in Figure 5.3 simulates M' using broadcasts. We will see that although such faithful simulations cannot always be ensured (simulating the zero-test is the problem), they can be ensured from some point on.

The controller process stores the current location of M' , while the users collectively store the values of counters. In a system instance of size $n + 1$, there are n user processes each of which is the product of two copies of the state diagram in Figure 5.3(c) (instantiated with $k = 1, 2$). Thus, the state of a user process is of the form (x_1, x_2) where the x_k s are the vertices pictured in Figure 5.3(c).

- If $x_k = 0_k$ then we say that the process' k th counter is set to zero;
- if $x_k = 1_k$, that its k th counter is set to one;
- if $x_k = disk$, that its k th counter is disabled; and
- if $x_k = t_k$ or t'_k , that its counter is in an intermediate state.

Ideally, the number of user processes with $x_k = 1$ is equal to the value of counter k .

In Figure 5.3 we see that increments and decrements are achieved as follows: to increment counter 1 the controller broadcasts this intention ($inc_k!!$) and goes to an intermediate state s_1

while the memory processes with k th counter set to zero (if any) respond and go to an intermediate state t_1 ; then exactly one memory process broadcasts that it has set its counter 1 bit from zero to one, and all the other processes, including the controller process, leave their intermediate state. It is an invariant of the simulation that if there is a user whose k th counter is in an intermediate state, say t_k , then (a) every process whose k th counter is in an intermediate state is in fact in state t_k , and (b) the controller process is also in an intermediate state, i.e., s_k (see Figure 5.3(b)). In this case the only possible next step is for some user process to broadcast an acknowledgement. Decrements are similar. On the other hand, if no process has a k th counter set to 0 (i.e., the memory storing the counter is full) and the controller broadcasts $inc_k!!$ then the simulation hangs. Similarly, if no process has a k th counter set to 1 and the controller broadcasts $dec_k!!$ then the simulation hangs. Thus, in an infinite run of a system instance, increments and decrements are faithfully simulated (\dagger), i.e., every time the controller tries to increment (or decrement) a counter, it does so successfully.

Zero transitions are more subtle—the controller may broadcast a “guess counter k is zero” command. If the k th counter of every user is set to zero, then the simulation proceeds faithfully, i.e., the control of the counter machine proceeds as if the counter is zero and indeed the total value of the counter stored in the users is zero; otherwise, every user whose k th counter is set to one enters a special state dis_k (thus permanently disabling it from being used as storage for counter k). In the second scenario the simulation is no longer faithful, i.e., the control of the counter machine proceeds as if the counter is zero but the value of the counter stored by the user processes is not zero. However, we will see that from some point on the simulation is faithful. To this end, let N_k be the number of processes whose k th counter is not disabled, and note that the sum $N_1 + N_2$ is non-increasing (\ddagger).

Now, if the 2CM M' enters the halting location m infinitely often, then the 2CM M has a (finite) run that enters the halting location m , and thus there is a run in a large enough system instance in which the control process enters m infinitely often.

Conversely, suppose that in some system instance there is a run such that the control process enters m infinitely often. By (\dagger), every time the controller tries to increment (or decrement) a counter, it does so successfully. By (\ddagger), from some point in time t onwards, the sum $N_1 + N_2$ is constant. Thus, at all later points $t' > t$ that the controller broadcasts “guess counter k is zero,” the k th counter of every process is equal to zero. Thus, the simulation after time t is faithful. Let $t'' \geq t$ be a time in which the controller is in the initial location. Then the run from this time onwards witnesses that M' enters the halting state m infinitely often.

In summary, the 2CM M halts if and only if there is a system instance that satisfies $\text{EGF}h_1$ where h_1 is the indexed atomic proposition that holds if the controller is in state m .

For the third item repeat the proof above except that initially the first scheduled process sends a broadcast message declaring itself a controller (it now acts as the controller and all the other processes act as users). \square

62 5. RENDEZVOUS AND BROADCAST

5.3.2 UNDECIDABILITY FOR ASYNCHRONOUS RENDEZVOUS

We discuss the undecidability results in the AR column of Figure 5.1. The proofs of the following undecidability results are similar and follow Emerson and Kahlon [2003b, Section 4].

Theorem 5.12 *Suppose synchronization is by asynchronous rendezvous. Then:*

1. $\text{PMCP}(\mathcal{P}, \mathbf{C}_\odot, \text{LTL}(C))$ is undecidable;
2. $\text{PMCP}(\mathcal{P}, \mathbf{C}_\odot, \omega\text{Reg}(\mathbf{A}))$ is undecidable; and
3. $\text{PMCP}(\mathcal{P}, \mathbf{C}_\odot, \forall U \text{ LTL}(U))$ is undecidable.

Proof Sketch. In the proof of Theorem 5.11 for $\text{LTL}(C)$, replace broadcast by asynchronous rendezvous. The only difference in behavior of the two simulations is that previously, if more than one process received a broadcast message, now, exactly one process receives the asynchronous-rendezvous message. The proof proceeds verbatim, except that one should note that at most one user is in an intermediate state at a time, and the phrase “every user whose k th counter is set to one enters a special state $disk_k$,” should change to “at least one user whose k th counter is set to one enters a special state $disk_k$.”

The case of $\omega\text{Reg}(\mathbf{A})$ is similar.

To extend this proof to the case $\forall U \text{ LTL}(U)$ we proceed differently from the broadcast case. The controller’s first move should be to synchronize, once and for all, with a single user, say v , and then to move into a special state where it is no longer used. On synchronizing the user v enters a special component of its state set and plays the role of the controller. The specification is of the form “for all users U , for all runs, if U initially synchronizes with the controller then it infinitely often satisfies h_1 . ” \square

5.4 DISCUSSION

In contrast to the token-passing systems of Chapter 4, the literature has focused on the more symmetric case of clique topologies for systems with communication by broadcast, pairwise rendezvous, or asynchronous rendezvous. One reason for this is that already in uni-directional rings the PMCP for these systems is undecidable, even for safety properties. This is because such systems can simulate a binary-valued token (as in Section 4.1.2 and Section 4.2.3).

However, even in a clique topology there are many undecidability results due to the fact that in a system of size n one can simulate the run of a 2CM as long as the counters do not exceed n . Common techniques to do this simulation include:

- using broadcast to elect a unique “controller” process (if needed) that simulates the control of the 2CM, while all the others are “memory” processes;

- using pairwise rendezvous between the controller and user processes to ensure that exactly one process increments/decrements the counter; and
- using broadcast (or asynchronous rendezvous) to reduce the number of active memory processes when the controller incorrectly “guesses” that a counter is zero.

We briefly discuss how the results of this chapter are affected under the assumption that there is no controller (i.e., all processes have the same type and are in a clique). Decidability proofs are not affected. Undecidability proofs in the case of no controller typically proceed as follows: first elect a controller using broadcast, and then proceed as in the case with a controller. Although broadcast is powerful enough to elect a controller, asynchronous rendezvous only seems powerful enough to elect a temporary controller.

Problem 5.13 Is the PMCP undecidable for systems communicating by asynchronous rendezvous, without a controller and for liveness specifications?

In summary, looking at Figure 5.1, all the entries (in the first three rows) are known to hold also for the case without controller, except for the columns AR and PR&AR and liveness specifications (i.e., the second and third rows).

Decidability results are typically proven using counter representations: since every process in a clique can communicate with every other process, it is enough to store the number of processes in every state. Pairwise rendezvous leads to a vector addition system with states, and thus certain safety and liveness properties are decidable. In contrast, broadcast and asynchronous rendezvous lead to well-structured transition systems, and thus safety is decidable, whereas it turns out that liveness is in general undecidable.

We note that the literature has only a few exact bounds on the complexity of the PMCP. We enumerate them here.

1. For broadcast protocols, [Schmitz and Schnoebelen \[2013\]](#) proved that the complexity of PMCP for coverability specifications is \mathbf{F}_ω -complete (the class \mathbf{F}_ω of Ackermannian problems is closed under primitive-recursive reductions).
2. For pairwise-rendezvous protocols:
 - (a) [Esparza \[2014, Section 3.3\]](#) reported that the complexity of PMCP for the coverability problem and pairwise rendezvous is EXPSPACE-complete.
 - (b) [Aminof et al. \[2014b\]](#) showed that PMCP is undecidable for pairwise rendezvous and 1-index $\text{CTL}^*\backslash X$ specifications.
 - (c) [Aminof et al. \[2014b\]](#) proved that for topologies that generalize cliques and stars (but not rings) that PMCP for 1-index $\text{LTL}\backslash X$ is EXPSPACE-complete (and PSPACE-complete without a controller), and that the program complexity (i.e., the formula is fixed) is EXPSPACE-complete (and in PTIME without a controller).

64 5. RENDEZVOUS AND BROADCAST

5.4.1 VARIATIONS OF THE MODEL

One variation are systems without a controller process. Here the connectivity graph is a clique, and all processes have the same type, i.e., all are user processes. The known results are summarized in Figure 5.1 (on page 52).

As a rule of thumb, systems with a unique controller are more difficult to verify than systems with only one process template, or systems with a constant number of process templates, but requiring that there is no bound on the number of processes instantiating each template. For instance, the PMCP for coverability specifications and pairwise-rendezvous synchronization is in PTIME if there is no controller (this follows immediately from German and Sistla [1992, Lemma 4.5]), while it is EXPSPACE-complete if there is a controller [Esparza, 2014, Section 3.3].

The proofs of decidability typically yield *cutoffs* that depend on both the template and the formula. For instance, for pairwise rendezvous with a controller, there is a cutoff (on the number of user processes) that is, roughly, doubly exponential in the size of the user template and the formula [German and Sistla, 1992, p. 687]. Since PMCP of liveness with broadcast is undecidable, there is no way to effectively compute cutoffs. However, the following restricted broadcast system has cutoffs for the specification language which is an extension of the universal fragment of $\text{CTL}^*\backslash X$ by the epistemic operator K , see Kouvaros and Lomuscio [2013a]: for every two local states q, q' and broadcast action label a , it holds that $(q, a!! , q')$ is a local transition if and only if $(q, a?? , q')$ is a local transition. Such a broadcast transition may be called *symmetric broadcast*. Systems that communicate with symmetric broadcast and pairwise rendezvous are, in the presence of a controller, quite expressive, e.g., Kouvaros and Lomuscio [2015] modeled a swarm aggregation algorithm. In the restricted case of a single symmetric broadcast action and pairwise rendezvous (or, more generally, k -wise rendezvous for some k), both safety and liveness are decidable [Aminof et al., 2015].

Ad hoc networks (Chapter 7) are concurrent systems in which the connectivity graphs are *general graphs* (not necessarily cliques), and the communication primitive is broadcast. Although the PMCP for general graphs and safety specifications is undecidable, one regains decidability by restricting the class of graphs (e.g., to graphs of bounded diameter).

The decidability results in the PR column of Figure 5.1 still hold if one only considers runs satisfying the following *fairness* condition: a run of a system instance is *fair* (following German and Sistla [1992, p. 680]) if every process that is enabled infinitely often is scheduled infinitely often. Here, a process is *enabled* in a global state if it can make an internal transition or synchronize with another process. However, the complexity is at least as hard as reachability for VASS.

The *Token-passing systems* that we see in Chapter 4 can be defined in terms of pairwise rendezvous, and there various fairness notions are used to get cutoff results. A similar “no-blocking” idea is used to get cutoffs for broadcast systems in the presence of a controller on cliques; see Kouvaros and Lomuscio [2013b, Definition 4.3].

CHAPTER 6

Guarded Protocols

In this chapter we extend the computational model of Section 2.2 with a new means of coordination. Until now processes coordinated with synchronized transitions and we have used transition labels to express which transitions should be taken together. In this chapter, we introduce guards, which are conditions on the global state and determine whether a specific local transition may be fired. We review results on systems with guards that contain quantifiers over all processes except the one evaluating the guard. Hence, the current state of other processes may restrict the control flow of a process. In contrast to synchronized transitions that, intuitively, link transitions of different processes, guards link transitions to the state of other processes.

Most of the models we discuss here contain two types of process templates, a single coordinator C , and user processes U . The systems are parameterized in the number of processes of type U .

The (un)decidability results we review here depend on the form of the guards as well as on the logic fragments used to formalize the specifications. Figure 6.1 and Table 6.1 give an overview of the results that we discuss in this chapter. In addition, in Section 6.7 we review restrictions of guards which ensure decidability, and are used to model cache-coherence protocols [Emerson and Kahlon, 2003a,c].

The undecidability results are proven by reduction to the non-halting problem of two-counter machines (cf. Section 3.1). The decidability proofs are based on the cutoff results by Emerson and Kahlon [2000], who construct runs in a small cutoff system from runs of bigger systems and vice versa.

6.1 MOTIVATING EXAMPLE

Consider a multi-threaded program, composed of n threads that are concurrently accessing or modifying a shared doubly linked list. Figure 6.2 shows an example of such a list. As is typical, we assume that a single update of a pointer in the list is atomic, that is, a thread reads an old pointer value, if another thread is currently writing to that pointer. However, our list is a non-atomic data structure, that is, while one process is updating several pointers in the list, the other processes may access the list. Figure 6.3 shows the list in an inconsistent state, when one thread started to delete the second element. To avoid such scenarios, we need a protocol that ensures that no other thread accesses the list in an inconsistent state. Such a protocol is usually called “multiple readers/single writer protocol.”

66 6. GUARDED PROTOCOLS

Table 6.1: Decidability results for Guarded Protocols and their sources, separated into systems with boolean guards ($\mathcal{P}_{\text{bool}}$), conjunctive guards ($\mathcal{P}_{\text{conj}}$), init-conjunctive guards ($\mathcal{P}_{\text{init-conj}}$), and disjunctive guards ($\mathcal{P}_{\text{disj}}$)

Result	Protocol	Graph	Specification	References
undecidability	$\mathcal{P}_{\text{bool}}$	clique (C)	$\text{LTL}(C)$	[Emerson and Namjoshi 1996] (proof idea)
undecidability	$\mathcal{P}_{\text{init-conj}}$	clique (C)	$\text{LTL}(C)$, $\text{Reg}(A)$, $\omega\text{Reg}(A)$	[Emerson and Kahlon 2003b]
a cutoff for $\ell \in \{1, 2\}$	$\mathcal{P}_{\text{init-conj}}[2]$	clique (C)	$1\text{-LTL}\backslash X(P^\ell)$, $1\text{-ELTL}\backslash X(P^\ell)$, $\text{LTL}\backslash X(C)$	[Emerson and Kahlon 2000]
a trivial cutoff for deadlock-free instances	$\mathcal{P}_{\text{init-conj}}$	clique (C)	as above	[Emerson and Kahlon 2000]
a trivial cutoff for finite-path properties	$\mathcal{P}_{\text{init-conj}}$	clique (C)	as above	[Emerson and Kahlon 2000]
a cutoff $\ell \in \{1, 2\}$	$\mathcal{P}_{\text{disj}}[2]$	clique (C)	$1\text{-LTL}\backslash X(P^\ell)$, $1\text{-ELTL}\backslash X(P^\ell)$, $\text{LTL}\backslash X(C)$	[Emerson and Kahlon 2000]
a cutoff	$\mathcal{P}_{\text{disj}}$	clique (C)	$2\text{-LTL}\backslash X(P^\ell, P^{\ell'})$, $2\text{-ELTL}\backslash X(P^\ell, P^{\ell'})$, $\text{LTL}\backslash X(C)$	[Emerson and Kahlon 2000]
decidability	$\mathcal{P}_{\text{disj}}$	clique (C)	$\text{Reg}(A)$, $\omega\text{Reg}(A)$	[Emerson and Kahlon 2003b]
decidability	$\mathcal{P}_{\text{disj}\&\text{rdvz}}$	clique (C)	$\text{Reg}(A)$, $\omega\text{Reg}(A)$, $\text{LTL}(C)$, $1\text{-LTL}\backslash X(P^\ell)$	[Emerson and Kahlon 2003b]
decidability	$\mathcal{P}_{\text{disj}\&\text{ardvz}}$	clique (C)	$\text{Reg}(A)$	[Emerson and Kahlon 2003b]
undecidability	$\mathcal{P}_{\text{disj}\&\text{ardvz}}$	clique (C)	$\omega\text{Reg}(A)$	[Emerson and Kahlon 2003b]
decidability	$\mathcal{P}_{\text{disj}\&\text{cbcast}}$	clique (C)	$\text{Reg}(A)$	[Emerson and Namjoshi 1998], [Emerson and Kahlon 2003b]
undecidability	$\mathcal{P}_{\text{disj}\&\text{cbcast}}$	clique (C)	$\omega\text{Reg}(A)$	[Esparza et al. 1999], [Emerson and Kahlon 2003b]
undecidability	$\mathcal{P}_{\text{conj}\&\text{disj}}$	clique (C)	all above	[Emerson and Kahlon 2003b], [Esparza et al. 1999]
undecidability	$\mathcal{P}_{\text{conj}\&[\text{a}]\text{rdvz}}$	clique (C)		
undecidability	$\mathcal{P}_{\text{conj}\&\text{cbcast}}$	clique (C)		

6.1. MOTIVATING EXAMPLE 67

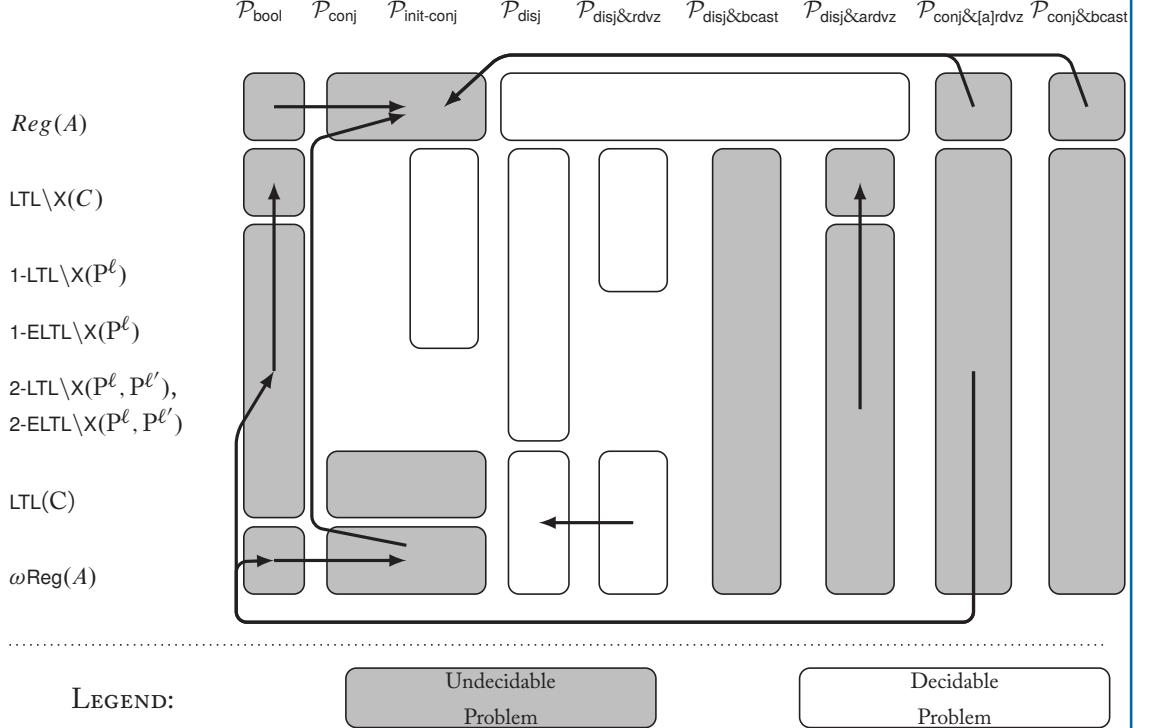


Figure 6.1: Decidability results for guarded protocols. An arrow from box A to box B means that the result of A follows from the result, or variation of the proof, of B . Blank entries are those not covered in the literature. The guarded protocols are defined over process templates P^1, \dots, P^d , and $1 \leq \ell, \ell' \leq d$. In case of $LTL \setminus X(C)$ and $LTL(C)$, we assume that template $C = P^1$, and there is only one process of type C .

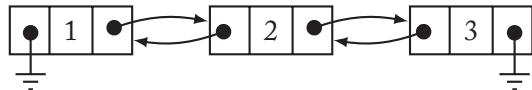


Figure 6.2: A doubly linked list shared by n threads.

To prevent several threads from simultaneously modifying the list or accessing the list that is being modified by another thread, we introduce a multiple readers/single writer protocol that is shown in Figure 6.4. The guards ϕ_1, \dots, ϕ_4 are defined in Equations 6.1–6.4. The formal syntax and semantics of guards follows in Section 6.2.

68 6. GUARDED PROTOCOLS

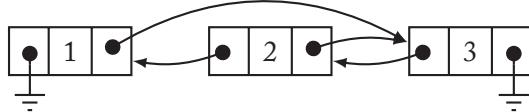


Figure 6.3: An inconsistent state of the list when one thread is deleting the second element.

$$\phi_1 \equiv [\forall \text{ other } j] \text{ neutral}_j \vee \text{try-write}_j \vee \text{try-read}_j \quad (6.1)$$

$$\phi_2 \equiv [\exists \text{ other } j] \text{ lock-write}_j \quad (6.2)$$

$$\phi_3 \equiv [\forall \text{ other } j] \text{ neutral}_j \vee \text{try-read}_j \vee \text{lock-read}_j \quad (6.3)$$

$$\phi_4 \equiv [\forall \text{ other } j] \text{ neutral}_j \vee \text{try-read}_j \vee \text{lock-read}_j \quad (6.4)$$

When a thread does not have to access or modify the list, it remains in the neutral state. When a thread has to modify the list, it changes its state to try-write. The thread can lock the list for exclusive access by changing its state to lock-write, provided that no other thread resides in the state lock-write, or in the state lock-read. The thread has an option to go back to the state neutral, if another thread has already entered the state lock-write. When the thread enters the state lock-write, it can iterate over the list and modify its elements; as soon as it finishes with the modifications, the thread goes back to the neutral state and thus allows other threads to access the list.

When a thread is going to iterate over the elements of the list without modifying it, the thread changes its state to try-read. The thread can lock the list for shared access by changing its state to lock-read, provided that there is no other thread in state try-write or lock-write, that is, no other thread has exclusively locked the list, or is trying to do so. Multiple processes can reside in the state lock-read and therefore can concurrently read the contents of the list. A thread must leave the state lock-read, as soon as it has finished reading its contents, or another thread has entered the state try-write.

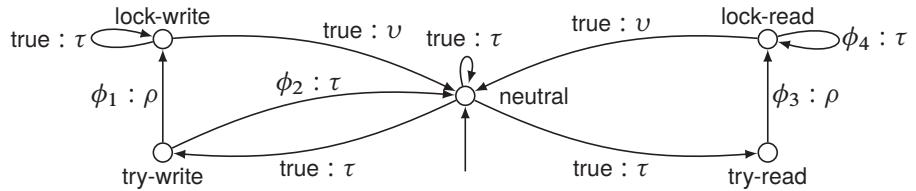


Figure 6.4: A process template modeling a multiple readers/single writer protocol. The circles represent local states labeled with state names, e.g., neutral. The edges represent transitions labeled with guards and actions, e.g., $\phi_1 : \rho$ for a guard ϕ_1 and an action ρ .

The protocol enforces the readers to have a lower priority than the writer. Moreover, the readers are preempted by a thread that is going to modify the list. When writes happen rarely,

multiple readers can efficiently access the list. On the other hand, when the writes occur frequently, the readers might starve, without ever accessing the list.

Note that, in order to lock the list, a thread has to make sure that there are no other threads in certain states. Although the exclusive access to the list can be enforced with a token-passing protocol (see Section 4), it is not easy to model the locking mechanism by multiple readers with token passing. The protocol can be modeled using broadcasts (see Section 5) by introducing auxiliary states and enforcing the threads to “listen” to the broadcasts by the threads locking and unlocking the list. This modeling, however, is closer to hardware rather than to multi-threaded programs, because the threads have to perform steps synchronously. Guarded protocols is a natural computational model for our example.

6.2 SYSTEM MODEL

To formalize guarded protocols, we slightly specialize the notion of process template, as we require specific atomic propositions to define guards. From this specialized process template, we obtain a system template in the usual way. Then, for a given system template we can define guards. In this way, we collect all ingredients to define guarded system instances: we associate to each *local* transition a guard, and restrict the *global* transition relation in a way that only admits global transitions that are built from local transitions whose guards evaluate to true in the current global state.

Identity-labeled process templates. We specialize the definition of a process template from Section 2.2 in that we identify atomic propositions with the local states, and restrict the label of a local state to be the singleton containing the state itself. That is, given d disjoint sets of local states Q^1, \dots, Q^d , we specialize AP_{pr} to be $Q^1 \cup \dots \cup Q^d$. An *identity-labeled process template* P^ℓ is a process template $P^\ell = (Q^\ell, Q_0^\ell, \Sigma_{\text{pr}}, \delta^\ell, \lambda^\ell)$, where for all $q \in Q^\ell$ it holds that $\lambda^\ell(q) = \{q\}$. Then, a tuple (P^1, \dots, P^d) of identity-labeled process templates is an *identity-labeled d-ary system template*.

Example 6.1 In our example in Figure 6.4, there is one process template P^1 with the set of local states $Q^1 = \{\text{neutral}, \text{try-read}, \text{try-write}, \text{lock-read}, \text{lock-write}\}$. The set of atomic propositions AP_{pr} equals to the set of local states Q^1 . The set of actions is $\Sigma_{\text{pr}} = \Sigma_{\text{int}} = \{\rho, \nu, \tau\}$. Every local state is labeled with itself, e.g., $\lambda^1(\text{neutral}) = \{\text{neutral}\}$.

Guards. We start by restricting the syntax of formulas that will be used in guards. In what follows, we will write $\phi(j)$ to denote a boolean formula over $\text{AP}_{\text{pr}} \times \{j\}$. In other words, $\phi(j)$ is a boolean formula over atomic propositions AP_{pr} of a process, indexed with a free index variable j . When j is instantiated with a constant $w \in V(G)$, the formula $\phi(w)$ becomes a boolean formula over AP_{sys} . Given $w \in V(G)$, the formula $\phi(w)$ is evaluated inductively at a

70 6. GUARDED PROTOCOLS

state $s = (q_1, \dots, q_{|V|}) \in S$ of a system instance \overline{P}^G as follows: for $p \in AP_{pr}$, it holds that $(\overline{P}^G, s) \models (p, w)$ if and only if $(p, w) \in \Lambda(s)$; The boolean connectives are interpreted as usual.

We define disjunctive, conjunctive, and boolean guards as follows.

- Let $\phi(j)$ be a disjunction over $AP_{pr} \times \{j\}$. Then,
 - $[\forall \text{ other } j] \phi(j)$ is a *conjunctive guard*, and
 - $[\exists \text{ other } j] \phi(j)$ is a *disjunctive guard*.
- Conjunctive and disjunctive guards are *boolean guards*. If f and g are boolean guards, then $\neg f$, $f \vee g$, and $f \wedge g$ are boolean guards as well.

To simplify notation, we write p_j to denote a proposition $(p, j) \in AP_{pr} \times \{j\}$.

Example 6.2 In the protocol in Figure 6.4, the guard $\phi_1 \equiv [\forall \text{ other } j] \text{ neutral}_j \vee \text{try-write}_j \vee \text{try-read}_j$ is an example of a conjunctive guard, whereas the guard $\phi_2 \equiv [\exists \text{ other } j] \text{ lock-write}_j$ is an example of a disjunctive guard.

One can see from the definition that the quantifiers $[\forall \text{ other } j]$ and $[\exists \text{ other } j]$ are never nested, but the boolean connectives can introduce subformulas with several quantifiers.

Intuitively, if one fixes a process v , then $[\forall \text{ other } j] \phi(j)$ means that ϕ should evaluate to true on *all processes* other than v , and $[\exists \text{ other } j] \phi(j)$ means that ϕ should evaluate to true on *at least one* process other than v . Formally, given a system instance \overline{P}^G , its global state s , a process $v \in V(G)$, and a guard f , we evaluate f in s with respect to v (in symbols, $(\overline{P}^G, s) \models_v f$) as follows.

- If f has the form $[\forall \text{ other } j] \phi(j)$, then $(\overline{P}^G, s) \models_v f$ if and only if for *every* process $w \in V(G) \setminus \{v\}$ it holds $(\overline{P}^G, s) \models \phi(w)$, where $\phi(w)$ is obtained from $\phi(j)$ by substituting all instances of j with w .
- Similarly, if f has the form $[\exists \text{ other } j] \phi(j)$, then $(\overline{P}^G, s) \models_v f$ if and only if there *exists* a process $w \in V(G) \setminus \{v\}$ with the property $(\overline{P}^G, s) \models \phi(w)$.
- If f is constructed using boolean connectives \neg , \vee , and \wedge , then $(\overline{P}^G, s) \models_v f$ is defined as usual via evaluation of the subformulas of f .

Consider a boolean formula $\text{all_states}(i) = \bigvee_{q \in Q^1 \cup \dots \cup Q^d} (q, i)$. Observe that the formula evaluates to true on every local state $q \in Q^1 \cup \dots \cup Q^d$ of a process i . Using $\text{all_states}(i)$, one can construct the conjunctive guard “[$\forall \text{ other } i$] $\text{all_states}(i)$ ” and the disjunctive guard “[$\exists \text{ other } i$] $\text{all_states}(i)$ ” that both evaluate to true on *every* global state. We abbreviate either of these guards with true: the form $[\forall \text{ other } i] \text{ all_states}(i)$ is used when the class of protocols is restricted to the protocols with conjunctive guards; the form $[\exists \text{ other } i] \text{ all_states}(i)$ is used when the class of protocols is restricted to the protocols with disjunctive guards.

Guarded protocols. A guarded protocol is an identity-labeled d -ary system template equipped with an assignment of a guard to each transition. Formally, a *guarded protocol* is a pair $((P^1, \dots, P^d), gd)$, where (P^1, \dots, P^d) is an identity-labeled d -ary system template with $P^\ell = (Q^\ell, Q_0^\ell, \Sigma_{pr}, \delta^\ell, \lambda^\ell)$, and gd maps every transition from $\bigcup_{1 \leq \ell \leq d} \delta^\ell$ to a guard.

Example 6.3 The protocol in Figure 6.4 has 10 transitions, each labeled with a guard. For instance, the transition $t = (\text{try-read}, \rho, \text{lock-read})$ is labeled with the guard ϕ_3 , or, formally, $gd(t) = \phi_3$.

When we consider guarded protocols with synchronization actions, a standard assumption is that receive actions are always enabled. More formally, for every $\text{in}_a \in AP_{pr}$ and every $t = (q, \text{in}_a, q') \in \delta^\ell$, we require that $gd(t) = \text{true}$. For instance, if in_a is a broadcast receive $a??$, then every transition with action label $a??$ is guarded by true.

Connectivity graphs. In this chapter we restrict d -ary connectivity graphs to cliques. Let $\mathbf{C} : \mathbb{N}^d \mapsto \mathbf{C}(\mathbf{n})$ be a sequence of graphs that associates a d -ary connectivity graph with a size vector $\mathbf{n} = (n_1, \dots, n_d) \in \mathbb{N}^d$. We require that each $\mathbf{C}(\mathbf{n})$ is a clique of size \mathbf{n} , i.e., if $\mathbf{C}(\mathbf{n}) = (V, E, \text{type})$, then $E = V \times V$ and for all $\ell \in [d]$, it holds $|\{v \in V : \text{type}(v) = \ell\}| = n_\ell$. We also introduce notation $\mathbf{I}(\mathbf{n})$ for the set of indices $\{i : 1 \leq i \leq n_1 + \dots + n_d\}$.

Example 6.4 The protocol in Figure 6.4 has one process template, that is, $d = 1$. Then, $\mathbf{C}(\mathbf{n}) : \mathbb{N} \rightarrow \mathbb{N}$ is a function that for each $n \geq 1$, returns a clique of size n , which corresponds to a system of n processes created from the template P^1 .

Guarded system instances. We construct a guarded system instance $\overline{P}^{\mathbf{C}(\mathbf{n})}$ of a guarded protocol $((P^1, \dots, P^d), gd)$ by restricting the global transition relation. In particular, we restrict internal and synchronous transitions—defined in Section 2.2—to guarded internal transitions and guarded synchronous transitions.

A *guarded internal transition* of $\overline{P}^{\mathbf{C}(\mathbf{n})}$ is a triple $(s, a, s') \in S \times \Sigma_{int} \times S$ for which there exists a process index $v \in V$ fulfilling the conditions (INT-STEP), (INT-FRAME), and

(INT-GUARD) The guard of the process transition $t_v = (s(v), a, s'(v))$ is satisfied in the global state s , that is, $(\overline{P}^{\mathbf{C}(\mathbf{n})}, s) \models_v gd(t_v)$.

A *guarded synchronous transition* is $(s, a, s') \in S \times \Sigma_{sync} \times S$ for which there exists a process index $v \in V$ and a set $\mathcal{I} \subseteq \{w \in V : E(v, w)\}$ of recipients of v in $\mathbf{C}(\mathbf{n})$ satisfying (CARD), (STEP), (I-STEP), (FRAME), (MAX), and the following condition:

(GUARD) The guard of the process transition $t_v = (s(v), out_a, s'(v))$ of process v is satisfied in the global state s , that is, $(\overline{P}^{\mathbf{C}(\mathbf{n})}, s) \models_v gd(t_v)$. (Recall that input actions are labeled with true.)

72 6. GUARDED PROTOCOLS

Finally, the global transition relation Δ of $P^{\mathbf{C}(n)}$ consists only of guarded internal and synchronous transitions.

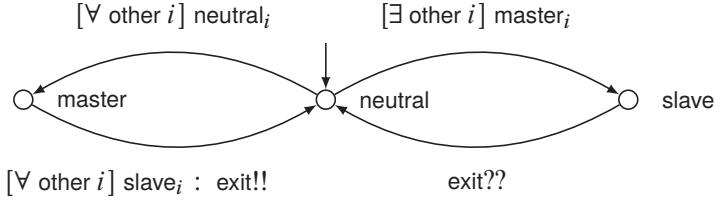


Figure 6.5: A barrier synchronization protocol. The first process enters the barrier as a master, and all other processes enter the barrier later as slaves. Once all processes gather in the states slave and master, the master broadcasts the action exit!! and all processes go to the state neutral synchronously. The guards of unlabeled edges are equal to true, and the actions of unlabeled edges are equal to τ .

Example 6.5 Consider a system instance $P_{RW}^{\mathbf{C}(3)}$ of three processes following the template P_{RW} that is shown in Figure 6.4. Let s and t be the states (neutral, try-read, try-read) and (neutral, lock-read, try-read), respectively. Then, the transition (s, τ, t) is a guarded internal transition of the system instance $(P_{RW})^{\mathbf{C}(3)}$. Indeed, the condition (INT-GUARD) is satisfied, as the guard ϕ_3 holds true in the global state s and therefore process 3 can perform its internal transition (try-read, τ , lock-read).

Example 6.6 Consider the process template P_B shown in Figure 6.5. The processes following the template P_B implement a barrier synchronization protocol: the processes enter the barrier and, as soon as all processes gather at the barrier, leave it synchronously. Let s and t be the states (master, slave, slave) and (neutral, neutral, neutral), respectively. Then, the transition (s, τ, t) is a guarded transition of the system instance $P_B^{\mathbf{C}(3)}$. Indeed, the condition (GUARD) is satisfied, as the guard $[\forall \text{other } i] \text{ slave}_i$ holds true for process 1 in the global state s , and therefore process 1 performs the transition (master, $x!!$, neutral). Moreover, according to the conditions (CARD) and (MAX), processes 2 and 3 must perform the transition (slave, $x??$, neutral).

Deadlocks. In this chapter, we require that the transition relation of a system instance is total. If there is a deadlock in a system, then the deadlocked run is extended to an infinite one with the deadlock state repeated at the end (cf. discussion in Section 2.2.2).

In contrast to most other systems considered in the literature, the parameterized deadlock detection problem for guarded protocols has been studied extensively, and decidability results for PMCP often extend to parameterized deadlock detection. This is the case for Theorems 6.22, 6.25, and 6.30 in this chapter.

6.2.1 CLASSES OF GUARDED PROTOCOLS

There are several important classes of guarded protocols in the literature. The following classes restrict the form of guards, and assume that only *internal transitions* occur, that is, $\Sigma_{sync} = \emptyset$:

\mathcal{P}_{bool} : guarded protocols with boolean guards [Emerson and Namjoshi, 1996];

\mathcal{P}_{disj} : guarded protocols using only disjunctive guards $[\exists \text{ other } j] \phi(j)$ [Emerson and Kahlon, 2000];

\mathcal{P}_{conj} : guarded protocols using only conjunctive guards $[\forall \text{ other } j] \phi(j)$ [Emerson and Kahlon, 2003b];

$\mathcal{P}_{init-conj}$: the subclass of \mathcal{P}_{conj} with the following restrictions [Emerson and Kahlon, 2000]:

- each process template P^ℓ has $init^\ell$ as the only initial state, i.e., $Q_0^\ell = \{init^\ell\}$; and
- every guard is satisfiable by the initial states, i.e., the guard can be written in the form $[\forall \text{ other } j] (init^1 \vee \dots \vee init^d \vee \psi(j))$.

Although both \mathcal{P}_{conj} and $\mathcal{P}_{init-conj}$ are usually called protocols with conjunctive guards, we differentiate $\mathcal{P}_{init-conj}$ from \mathcal{P}_{conj} , as they differ in decidability. In particular, the former have remarkable decidability results.

Example 6.7 Consider the guarded protocol (P, gd) shown in Figure 6.4. As it contains both conjunctive and disjunctive guards, the protocol belongs to the class \mathcal{P}_{bool} . Let (P', gd') be the protocol obtained from (P, gd) by removing the transition that is labeled with ϕ_2 . Then, the resulting protocol belongs to the class \mathcal{P}_{conj} . Moreover, as the guards ϕ_1, ϕ_3 , and ϕ_4 contain the initial state neutral, the guarded protocol (P', gd') belongs to the class $\mathcal{P}_{init-conj}$.

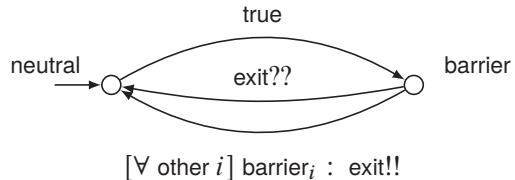


Figure 6.6: A simplified version of the barrier synchronization protocol shown in Figure 6.5. Every process may enter the state barrier. Once all processes gather in the state barrier, one of them broadcasts the action `exit!!` and all processes go to the state `neutral` synchronously. The guards of unlabeled edges are equal to `true`, and the actions of unlabeled edges are equal to τ .

Further, when synchronous transitions are allowed, we consider the classes of the form $\mathcal{P}_{g\&z}$. Each such a class contains guards of type $g \in \{\text{conj}, \text{disj}\}$, internal guarded transitions, and guarded transitions with the synchronization primitives of type $z \in \{\text{bcast}, \text{rdv}, \text{ardv}\}$, where conj

74 6. GUARDED PROTOCOLS

and disj stand for conjunctive and disjunctive guards, while bcast , rdv , and ardv denote broadcast, pairwise rendezvous, and asynchronous rendezvous, respectively (Section 2.2.1). For instance, $\mathcal{P}_{\text{disj}\&\text{bcast}}$ and other combinations of disjunctive guards with synchronization primitives have been investigated by Emerson and Kahlon [2003a,b,c]. We revisit the results on combinations of guards with synchronization primitives in Section 6.6.

Example 6.8 The guarded protocol shown in Figure 6.5 belongs to the class $\mathcal{P}_{\text{bool}\&\text{bcast}}$. Figure 6.6 shows a simplified version of the barrier protocol that belongs to the class $\mathcal{P}_{\text{conj}\&\text{bcast}}$.

In what follows, by $\mathcal{P}[\mathbf{d}]$ we denote guarded protocols from class \mathcal{P} with exactly \mathbf{d} process templates.

One can find two syntactically different forms of disjunctive guards in the literature. The first paper on guarded systems [Emerson and Namjoshi, 1996]¹ introduces disjunctive guards as expressions $[\exists \text{ other } j] \phi(j)$, where $\phi(j)$ is a boolean formula over $\text{AP}_{\text{pr}} \times \{j\}$. Recall that in this book we consider $\phi(j)$ only in restricted form, namely, $\phi(j)$ is a disjunction over propositions from $\text{AP}_{\text{pr}} \times \{j\}$; cf. Emerson and Kahlon [2000, 2003a,b,c]. In fact, both definitions of guarded systems have the same expressive power. Indeed, every global state of a system instance is labeled with exactly one local state of every process, and thus it is easy to prove the following observation.

Observation 6.9 Consider a guarded d -ary system template (P^1, \dots, P^d) . Let $\phi(j)$ be a boolean formula over $\text{AP}_{\text{pr}} \times \{j\}$. Then there exists a formula $\psi(j) = p_1 \vee \dots \vee p_k$ with $\{p_1, \dots, p_k\} \subseteq \text{AP}_{\text{pr}} \times \{j\}$ with the following property:

For every clique $\mathbf{C}(n)$, for every global state s of system instance $\overline{P}^{\mathbf{C}(n)}$, and for every process $v \in V(\mathbf{C}(n))$, it holds $\overline{P}^{\mathbf{C}(n)}, s \models \phi(v)$ if and only if $\overline{P}^{\mathbf{C}(n)}, s \models \psi(v)$ is true.

6.2.2 SPECIFICATIONS

Indexed formulas on guarded protocols restrict the scope of indexed variables to a single process template. Recall that the notation $\forall i : \text{type}(i) = \ell. \phi(i)$ denotes that i ranges only over the indices of processes instantiated from template P^ℓ .

In what follows, we consider several classes of specifications met in the literature.

- Classes $1\text{-LTL}\backslash X(P^\ell)$ and $1\text{-ELTL}\backslash X(P^\ell)$ contain *one-index* formulas over the atomic propositions of processes having the same fixed process template P^ℓ . If $\phi(i)$ is an $\{A, E\}$ -free $1\text{-CTL}^*\backslash X$ path formula over $Q^\ell \times \{i\}$, then “ $\forall i : \text{type}(i) = \ell. A\phi(i)$ ” and “ $\forall i : \text{type}(i) = \ell. E\phi(i)$ ” are $1\text{-LTL}\backslash X(P^\ell)$ and $1\text{-ELTL}\backslash X(P^\ell)$ formulas, respectively.²

¹We do not include the results by Emerson and Namjoshi [1996] in this book, as they consider synchronous systems, that is, the systems where all processes make a step at once.

²In the nonparameterized case, one can reduce the problem of checking an ELTL-formula $E\neg\psi$ on a Kripke structure M to $M \not\models A\psi$. This, however, does not work in the parameterized case: a straightforward negation of PMCP for $1\text{-LTL}\backslash X(P^\ell)$, that is, $\neg(\forall n \geq (1, \dots, 1). \overline{P}^n \models \forall i : \text{type}(i) = \ell. A\phi(i))$ is not equivalent to PMCP for $1\text{-ELTL}\backslash X(P^\ell)$, that is, $\forall n \geq (1, \dots, 1). \overline{P}^n \models \forall i : \text{type}(i) = \ell. E\neg\phi(i)$.

- Classes $\text{LTL}(C)$ and $\text{LTL}\backslash X(C)$ contain LTL and $\text{LTL}\backslash X$ formulas, respectively, over the states of the sole controller process of type 1. This only process is created from a process template P^1 and is usually called the *controller*, while the other processes are created from a process template P^2 and are usually called the *users*. In what follows, we will use C to denote the controller process template P^1 and U to denote the user process template P^2 . One can see $\text{LTL}\backslash X(C)$ as a subclass of $1\text{-LTL}\backslash X(C)$, where the process type is compared to 1, that is, a specification is of the form $\forall i: \text{type}(i) = 1. \varphi(i)$, and there is only one process of type 1. The properties from these classes are also discussed in Section 5.
- Classes $2\text{-LTL}\backslash X(P^\ell, P^{\ell'})$ and $2\text{-ELTL}\backslash X(P^\ell, P^{\ell'})$ are similar to $1\text{-LTL}\backslash X(P^\ell)$ and $1\text{-LTL}\backslash X(P^\ell)$, but they contain *two-index* formulas over the states of processes created from process templates P^ℓ and $P^{\ell'}$. That is, the formulas have the form
 - $\forall i: \text{type}(i) = \ell. (\forall j: \text{type}(j) = \ell'. A\phi(i, j))$ or
 - $\forall i: \text{type}(i) = \ell. (\forall j: \text{type}(j) = \ell'. E\phi(i, j))$.
- Classes $\text{Reg}(A)$ and $\omega\text{Reg}(A)$ are regular and ω -regular languages over actions Σ_{int} as defined in Section 2.4.2.

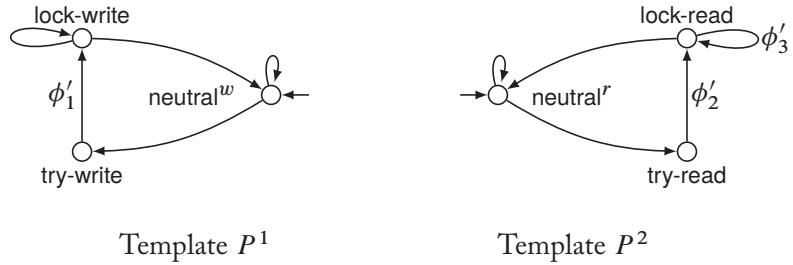


Figure 6.7: Another modeling of multiple readers/single writer protocol using two process templates: template P^1 models a writer in the system; template P^2 models a reader in the system. The guards ϕ'_1 , ϕ'_2 , and ϕ'_3 are defined in Equations 6.5–6.7. The guards of unlabeled edges are equal to true, and the actions of unlabeled edges are equal to τ .

Example 6.10 Consider the guarded protocol shown in Figure 6.4.

The specification $\forall i: \text{type}(i) = 1. AG(\text{try-write}_i \rightarrow F(\text{lock-write}_i \vee \text{neutral}_i))$ states that in every execution, every process that is trying to obtain the write lock will eventually do so, or will fall back to the state neutral . This specification belongs to the class $1\text{-LTL}\backslash X(P^1)$.

The specification $\forall i: \text{type}(i) = 1. EG(\text{try-write}_i \rightarrow F \text{ lock-write}_i)$ states that there is at least one execution, where every process that is trying to obtain the write lock will do so. This specification belongs to the class $1\text{-ELTL}\backslash X(P^1)$. The specification $(\rho \tau^* v)^*$ states that locking action ρ

76 6. GUARDED PROTOCOLS

and unlocking action ν alternate; the specification belongs to the class $\text{Reg}(A)$. Similarly, the specification $(\rho \tau^* \nu)^\omega$ states that locking and unlocking actions alternate infinitely often; it belongs to the class $\omega\text{Reg}(A)$.

Example 6.11 Consider the guarded protocol shown in Figure 6.7. The guards ϕ'_1 , ϕ'_2 , and ϕ'_3 are defined as follows:

$$\phi'_1 \equiv [\forall \text{ other } j] \text{neutral}_j^r \vee \text{neutral}_j^w \vee \text{try-read}_j \vee \text{try-write}_j \quad (6.5)$$

$$\phi'_2 \equiv [\forall \text{ other } j] \text{neutral}_j^r \vee \text{neutral}_j^w \vee \text{try-read}_j \vee \text{lock-read}_j \quad (6.6)$$

$$\phi'_3 \equiv [\forall \text{ other } j] \text{neutral}_j^r \vee \text{neutral}_j^w \vee \text{try-read}_j \vee \text{lock-read}_j \quad (6.7)$$

The specification $\forall i: \text{type}(i) = 1. (\forall j: \text{type}(j) = 2. \text{AG}(\neg \text{lock-write}_i \vee \neg \text{lock-read}_j))$ states that in every execution, no reader and writer can lock the critical section simultaneously. This specification belongs to the class $2\text{-LTL}\backslash X(P^1, P^2)$.

The specification $\forall i: \text{type}(i) = 1. (\forall j: \text{type}(j) = 2. \text{EG}((\text{try-write}_i \wedge \text{lock-read}_j) \rightarrow \text{lock-write}_i))$ states that there is an execution, in which a writer eventually obtains the lock, when a reader holds a lock. This specification belongs to the class $2\text{-ELTL}\backslash X(P^1, P^2)$.

Example 6.12 Consider the guarded protocol shown in Figure 6.7. Denote with C process template P^1 and with U process template P^2 . Further, assume that every system instance contains exactly one instance of P^1 .

The specification $G(\text{try-write} \rightarrow F \text{ lock-write})$ states that in every execution, the controller—the only writer in the system—eventually obtains the write lock, whenever it tries to enter the critical section. This specification belongs to the classes $LTL\backslash X(C)$ and $LTL(C)$.

The specification $G(\text{try-write} \rightarrow X(\text{lock-write} \vee X \text{ lock-write}))$ states that in every execution the controller obtains the write lock in at most two system steps, after it has entered the state try-write (this specification is violated on a system instance with at least three readers). This specification belongs to the class $LTL(C)$, but not to the class $LTL\backslash X(C)$.

6.3 UNDECIDABILITY: BOOLEAN AND CONJUNCTIVE GUARDS

We start by showing that PMCP with boolean guarded protocols is undecidable in Theorem 6.13. For conjunctive guarded protocols, we review results from Emerson and Kahlon [2003b, Propositions 3.1, 3.2, 3.3] that show that PMCP is undecidable for $\text{Reg}(A)$, $\omega\text{Reg}(A)$, and $LTL(C)$. We will see that the same proofs apply to the class $\mathcal{P}_{\text{init-conj}}$.

Theorem 6.13 $\text{PMCP}(\mathcal{P}_{\text{bool}}, C, LTL(C))$ is undecidable. The non-halting problem for two-counter machines can be reduced to $\text{PMCP}(\mathcal{P}_{\text{bool}}, C, \{G\neg holt_C\})$ with $d = 2$, where $holt_C$ is a local state of the controller C .

Proof. This proof is based on the idea mentioned by Emerson and Namjoshi [1996] in the conclusions.

Recall the notation from Section 3.1. Let \mathcal{M} be a 2CM with locations $[m]$, the initial state 1, the halting state m , and a set of commands $\Delta \subseteq [m] \times \mathcal{A} \times [m]$. We denote the two counters of \mathcal{M} by A and B . Given such a machine, we construct two process templates: Template C for the coordinator emulating the control flow of Δ ; Template U for the user processes. Each user process is either storing a single unary digit of A or B , or staying in the standby state $init_U$. Given $N \geq 1$, the system $(C, U)^{C(2N+1)}$ simulates at least N steps of \mathcal{M} .

The construction ensures that whenever the controller of type C begins to execute a command a , exactly one user process of type U performs a unary increment or decrement as prescribed by a . This is ensured by the controller and the user processes using conjunctive and disjunctive guards as follows.

- Using either a conjunctive or a disjunctive guard, the controller tests the counters for zero by observing the local states of the user processes.
- The controller tests with a disjunctive guard, whether at least one user process heard the command by C and started to execute it.
- A user process tests with a conjunctive guard that no other user process has started to execute the command issued by the controller.

We construct template C such that it has three kinds of states.

- The initial state $init_C$.
- A state $i \in [m]$ for each location i of the counter machine.
- A state $\langle i, a, i' \rangle$ representing a transition $(i, a, i') \in \Delta$. We need this state to record that the machine is in the middle of executing a command $a \in \{inc(A), dec(A), inc(B), dec(B)\}$. When C moves from $\langle i, a, i' \rangle$, the system has finished to simulate the command a at location i and is continuing with the command at location i' .

Template U has the following states.

- The initial state $init_U$, where the process contributes neither to A , nor to B .
- Storage states A_1 and B_1 reflecting that the process contributes a unary 1 to A and B , respectively.
- Temporary states A_{01} and B_{01} meaning that the process is in the middle of adding a unary 1 to A or B , respectively.
- Temporary states A_{10} and B_{10} meaning that the process is in the middle of subtracting a unary 1 from A or B , respectively.

78 6. GUARDED PROTOCOLS

Table 6.2: The guards of the transitions that simulate the commands over counter A . The sets Δ_A^+ and Δ_A^- are defined as follows: $\Delta_A^+ = \{(i, a, i') \in \Delta \mid a = inc(A)\}$ and $\Delta_A^- = \{(i, a, i') \in \Delta \mid a = dec(A)\}$. To obtain the case for B , swap A and B

C's transition	Guard
$(init_C, \tau, 1)$	$[\forall \text{ other } j] init_U$
$(i, inc(A), \langle i, inc(A), i' \rangle)$	$[\forall \text{ other } j] init_U \vee A_1 \vee B_1$
$(i, dec(A), \langle i, dec(A), i' \rangle)$	$[\forall \text{ other } j] init_U \vee A_1 \vee B_1$
$(\langle i, inc(A), i' \rangle, end(A), i')$	$[\exists \text{ other } j] A_{01}$
$(\langle i, dec(A), i' \rangle, end(A), i')$	$[\exists \text{ other } j] A_{10}$
$(i, zero(A), i')$	$[\forall \text{ other } j] init_U \vee B_1$
U's transition	Guard
$(init_U, inc(A), A_{01})$	$[\forall \text{ other } j] init_U \vee A_1 \vee B_1 \vee \bigvee_{\substack{t \in \Delta_A^+ \\ m}} \langle t \rangle$
(A_{01}, τ, A_1)	$[\forall \text{ other } j] init_U \vee A_1 \vee B_1 \vee \bigvee_{k=1}^m k$
$(A_1, dec(A), A_{10})$	$[\forall \text{ other } j] init_U \vee A_1 \vee B_1 \vee \bigvee_{\substack{t \in \Delta_A^- \\ m}} \langle t \rangle$
$(A_{10}, \tau, init_U)$	$[\forall \text{ other } j] init_U \vee A_1 \vee B_1 \vee \bigvee_{k=1}^m k$

Figure 6.8 depicts the transitions of template C , and Figure 6.9 depicts the transitions of template U . Table 6.2 assigns the guards to the transitions of C and U .

When a user process observes at least one process at state $\langle i, a, i' \rangle$ —this can be only the controller—it starts to execute the command a by moving to one of temporary states $\{A_{01}, A_{10}, B_{01}, B_{10}\}$. After that the controller waits until one of U 's leaves its intermediate state. To prevent several user processes from executing the same command, the transition guards ensure that no other user process has started to execute the command. For instance, two user processes cannot both move to A_{01} after seeing $\langle i, inc(A), i' \rangle$, because as soon as the first of them moved to A_{01} , the guard of the second process becomes false, as the first process violates $[\forall \text{ other } j] init_U \vee A_1 \vee B_1 \vee \bigvee_{\langle i, inc(A), i' \rangle \in \Delta} \langle i, inc(A), i' \rangle$. Note that we are exploiting the interleaving semantics in this argument.

Given $N \geq 1$, the system instance of $2N$ processes of type U and one process of type C simulates at least N steps of the two counter machine \mathcal{M} . By identifying proposition $halt_C$ with the state m of the controller, we specify the non-halting property as $G \neg m$ over the state of C . As we have encoded the undecidable non-halting problem of an arbitrary two-counter machine \mathcal{M} as $\text{PMCP}(\{C, U\}, \mathbf{C}, (G \neg m))$, we have proven the claim. \square

From Theorem 6.13, we know that $\text{PMCP}(\mathcal{P}_{\text{bool}}, \mathbf{C}, \text{LTL}(C))$ is undecidable. So, we immediately have undecidability for the following specification classes.

- As the LTL-formula in the proof does not use the next-time operator X , the proof applies to $\text{LTL} \setminus \text{X}(\mathcal{C})$ without modification.
- As $\text{LTL} \setminus \text{X}(\mathcal{C})$ is subclass of $2\text{-LTL} \setminus \text{X}(\mathcal{P}^\ell, \mathcal{P}^{\ell'})$, the proof applies to $2\text{-LTL} \setminus \text{X}(\mathcal{P}^\ell, \mathcal{P}^{\ell'})$ and $2\text{-ELTL} \setminus \text{X}(\mathcal{P}^\ell, \mathcal{P}^{\ell'})$ as well.
- One can encode a non-deterministic choice whether a process takes the role of the controller or the user. Using conjunctive guards one assures that the first process to take a step becomes the controller, and all other processes become user processes. Then Theorem 6.13 applies to $1\text{-LTL} \setminus \text{X}(\mathcal{P}^\ell)$ and $1\text{-ELTL} \setminus \text{X}(\mathcal{P}^\ell)$.

Corollary 6.14 $\text{PMCP}(\mathcal{P}_{\text{bool}}, \mathbf{C}, \mathcal{F})$ is undecidable for every specification class \mathcal{F} from the list: $\text{LTL} \setminus \text{X}(\mathcal{C})$, $\text{LTL}(\mathcal{C})$, $1\text{-LTL} \setminus \text{X}(\mathcal{P}^\ell)$, $1\text{-ELTL} \setminus \text{X}(\mathcal{P}^\ell)$, $2\text{-LTL} \setminus \text{X}(\mathcal{P}^\ell, \mathcal{P}^{\ell'})$, $2\text{-ELTL} \setminus \text{X}(\mathcal{P}^\ell, \mathcal{P}^{\ell'})$.

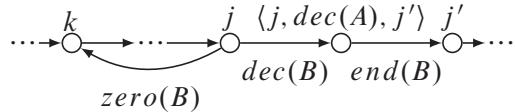
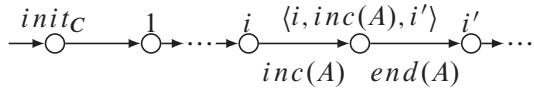


Figure 6.8: Template C simulating the control flow of a 2CM over counters A and B . The edges are labeled with actions; if an edge is not labeled, then the action is τ .

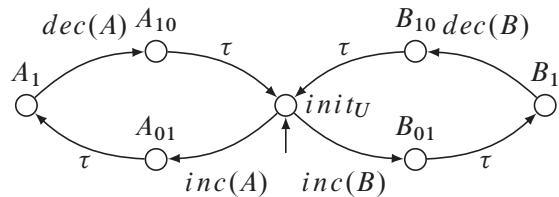


Figure 6.9: Template U of user processes that together simulate two integer counters A and B . An instance of U contributes a unary 1 to A (resp. B), when in A_1 (resp. in B_1). The edges are labeled with actions; if an edge is not labeled, then the action is τ .

We now turn our attention to conjunctive guards. The case of conjunctive guards is undecidable for $\text{LTL}(\mathcal{C})$, as shown by Emerson and Kahlon [2003b, Propositions 3.1–3.2]. We re-use the

80 6. GUARDED PROTOCOLS

Table 6.3: The guards of the transitions simulating the 2CM's commands labeled with actions $a \in \{inc(A), dec(A)\}$ and $zero(A)$. To obtain the case for B , swap A and B

C's transition	Guard
$(init_C, \tau, 1)$	$[\forall \text{ other } j] init_U$
$(i, a, \langle i, a, i' \rangle)$	$[\forall \text{ other } j] init_U \vee A_1 \vee B_1$
$(\langle i, a, i' \rangle, end(A), i')$	$true$
$(i, zero(A), i')$	$[\forall \text{ other } j] init_U \vee B_1$
U's transition	Guard
$(init_U, inc(A), A_{01})$	$[\forall \text{ other } j] init_U \vee A_1 \vee B_1 \vee \bigvee_{t \in \Delta_A^+} \langle t \rangle$
$(A_1, dec(A), A_{10})$	$[\forall \text{ other } j] init_U \vee A_1 \vee B_1 \vee \bigvee_{t \in \Delta_A^-} \langle t \rangle$
(A_{01}, τ, A_1)	$[\forall \text{ other } j] init_U \vee A_1 \vee B_1 \vee \bigvee_{k=1}^m k$
$(A_{10}, \tau, init_U)$	$[\forall \text{ other } j] init_U \vee A_1 \vee B_1 \vee \bigvee_{k=1}^m k$

construction of C and U from the proof of Theorem 6.13. As we cannot use disjunctive guards anymore, we replace several guards with $true$ as reflected in Table 6.3. Rather, we encode the interlocking behavior in the specification using the next-time operator \mathbf{X} .

Theorem 6.15 $\text{PMCP}(\mathcal{P}_{\text{conj}}, \mathbf{C}, \text{LTL}(C))$ is undecidable. The non-halting problem for two-counter machines can be reduced to $\text{PMCP}(\mathcal{P}_{\text{conj}}, \mathbf{C}, \{\mathbf{G}\neg halt_C\})$ with $d = 2$, where $halt_C$ is a local state of the controller C .

Proof. We change the commands from Table 6.2 of the proof of Theorem 6.13, as shown in Table 6.3. The guards of the transitions $(\langle i, inc(A), i' \rangle, end(A), i')$ and $(\langle i, dec(A), i' \rangle, end(A), i')$ are replaced with $true$. With the new guards the reachability of location m in a family $\{(C, U)^{\mathbf{C}(1,n)}\}_{n \geq 1}$ does not simulate the two-counter machine anymore, as the process C may invoke $inc(A)$ and then immediately proceed to the next control location without waiting for a user process performing the operation. Indeed, the guard is $true$, so C can jump out of $\langle i, inc(A), i' \rangle$ earlier than it could with the disjunctive guards in Table 6.2. To deal with that issue, we only consider restricted runs. This can be done by constructing a specification that uses the \mathbf{X} operator, and is thus represented with $\text{LTL}(C)$ instead of $\text{LTL} \setminus \mathbf{X}(C)$. Formally, given an action $a \in \{inc(A), dec(A), dec(B), dec(B)\}$ of the counter machine, we introduce an LTL formula $yield_a$:

$$yield_a = \mathbf{G} \left(\bigwedge_{(i,a,i') \in \Delta} ((\neg \langle i, a, i' \rangle \wedge \mathbf{X} \langle i, a, i' \rangle) \rightarrow \mathbf{XX} \langle i, a, i' \rangle) \right).$$

Observe that in runs in which $yield_{inc(A)}$ holds, C cannot make the next step immediately after going to $\langle i, inc(A), i' \rangle$. As a consequence, the formula can be used to filter out only the executions, where a process of type U makes a step right after C moved to $\langle i, inc(A), i' \rangle$.

Finally, the $yield_a$ formulas are combined with the non-halting property. Thus, we arrive at the parameterized model checking problem, where a system instance with $2N$ user processes is simulating at least N steps of the two counter machine \mathcal{M} :

$$\forall n \geq 1. (C, U)^{\mathbf{C}^{(1,n)}} \models (yield_{inc(A)} \wedge yield_{inc(B)} \wedge yield_{dec(A)} \wedge yield_{dec(B)}) \rightarrow G(\neg \text{halt}_C).$$

As this problem is an instance of $\text{PMCP}(\mathcal{P}_{\text{conj}}, \mathbf{C}, \text{LTL}(C))$, we conclude that the latter is undecidable.

Note carefully that the constructed specification is not a safety property, as its negated normal form is using temporal operators G and F . \square

In fact, the guarded protocol constructed in the proof of Theorem 6.15 uses only init-conjunctive guards, that is, all the guards in Table 6.3 contain $init_U$. Thus, we immediately arrive at the conclusion that PMCP is undecidable for $\mathcal{P}_{\text{init-conj}}$, which is a subclass of $\mathcal{P}_{\text{conj}}$:

Corollary 6.16 *The problem $\text{PMCP}(\mathcal{P}_{\text{init-conj}}, \mathbf{C}, \text{LTL}(C))$ is undecidable.*

As noted by Emerson and Kahlon [2003b, Proposition 3.3], the proof of Theorem 6.15 can be also applied to regular and ω -regular action-based specifications.

Theorem 6.17 *$\text{PMCP}(\mathcal{P}_{\text{conj}}, \mathbf{C}, \text{Reg}(A))$ and $\text{PMCP}(\mathcal{P}_{\text{conj}}, \mathbf{C}, \omega\text{Reg}(A))$ are undecidable.*

Proof. We re-use the proof from Theorem 6.15, except that the $yield$ constraints have to be expressed as a regular language over actions, rather than an LTL formula. We show how to express $yield_{inc(A)}$ and $yield_{dec(A)}$; the formulas for B are obtained similarly.

Let letter i denote an action $inc(A)$ or $dec(A)$ of C , letter e denote the action $end(A)$, and expression X be the regular expression that recognizes the complement $\Sigma_{pr} \setminus \{i, e\}$ of i and e . Then $yield_i$ can be written as the following regular expression:

$$yield_i = (iie \mid X)^*.$$

As the parameterized model checking problem is undecidable for regular properties, so it is for ω -regular properties. \square

6.4 DECIDABILITY: INIT-CONJUNCTIVE AND DISJUNCTIVE GUARDS

In contrast to the classes of boolean and conjunctive guards of the previous section, the PMCP over indexed $\text{LTL} \setminus X$ formulas and $\text{LTL} \setminus X(C)$ is decidable for the classes of init-conjunctive and disjunctive guards [Emerson and Kahlon, 2000]. In this section, we present the results from a

82 6. GUARDED PROTOCOLS

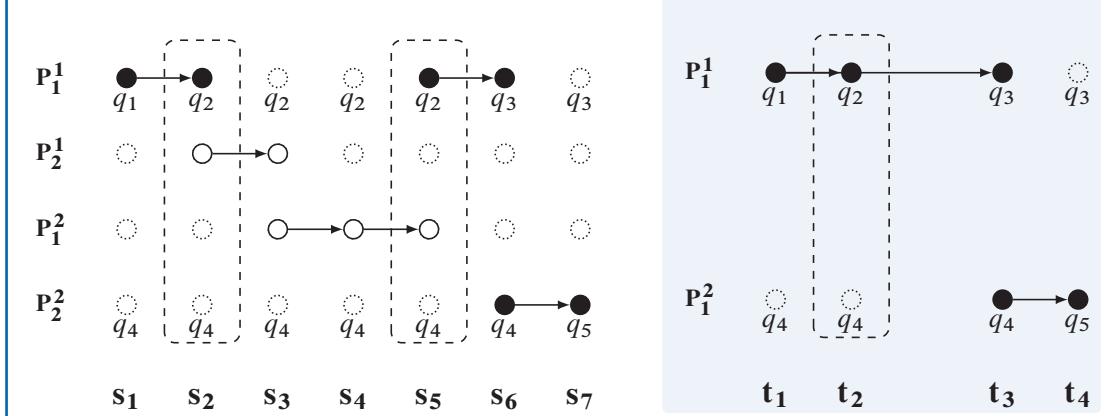


Figure 6.10: Illustration of path reduction. Processes 1 and 4 of $(P^1, P^2)^{(2,2)}$ are projected onto Processes 1 and 2 of $(P^1, P^2)^{(1,1)}$ using $pr_{idx} : \{1 \mapsto 1, 4 \mapsto 2\}$. Thus, the run $s_1 \dots s_7$ becomes the sequence $t_1 \dots t_4$ by collapsing $s_2 \dots s_5$ to t_2 and hiding the local states of processes 2 and 3 on the left. Observe that $t_1 = pr_{st}(s_1)$ and $t_4 = pr_{st}(s_7)$.

slightly different angle than the original work by Emerson and Kahlon [2000]: we give a general proof schema that structures cutoff proofs for the disjunctive and init-conjunctive guards; and we explain why both types of guards preserve their satisfiability when the system has “enough” processes. We have to pay attention to deadlocked computations in all cases. Here, we give the proofs of the cutoff results as instances of two general proof schemas: namely, *bounding* shows how to reduce a run in a “large” system to a run in “small” cutoff system; *monotonicity* constructs an equivalent run (a term we make precise later) of a large system from a run of a small system. Together, bounding and monotonicity prove equivalence of large systems to the cutoff system with respect to the formulas from $1\text{-LTL}\backslash X$, $1\text{-ELTL}\backslash X(P^\ell)$, and $\text{LTL}\backslash X(C)$.

6.4.1 PRELIMINARIES

We start with the ingredients of the schemas: the counting function, path reduction, and symmetry argument.

Counting function. Given a system instance $(P^1, P^2)^{\mathbf{C}(n_1, n_2)}$ with the set of global states S , we define the *counting function* $\kappa : S \times (Q^1 \cup Q^2) \rightarrow \mathbb{N}_0$. For every global state $s \in S$ and a local state $q \in Q^1 \cup Q^2$, the number of times q occurs in s is given by $\kappa(s, q)$. When it is clear from context, we apply the same symbol κ to different system instances.

Path reduction. Intuitively, given a path (a sequence of global states) in a large system, we project the global states onto a subset of the processes. Steps in the original path performed by the pro-

cesses that are not in the subset result in stuttering in the projection. In path reduction we remove stuttering that is due to this effect. See Figure 6.10 for an illustration.

Formally, consider two system instances $(P^1, P^2)^{\mathbf{C}(n_1, n_2)}$ and $(P^1, P^2)^{\mathbf{C}(m_1, m_2)}$ with $n_1 \geq m_1$ and $n_2 \geq m_2$. An *index projection* is a bijective partial function pr_{idx} from $V(\mathbf{C}(n_1, n_2))$ to $V(\mathbf{C}(m_1, m_2))$. For $v \in V(\mathbf{C}(n_1, n_2))$, we write $pr_{idx}(v) = \perp$ to denote that pr_{idx} is undefined on v . Using pr_{idx} , we define a state projection pr_{st} . Namely, given a state s of $(P^1, P^2)^{\mathbf{C}(n_1, n_2)}$, state $t = pr_{st}(s)$ of $(P^1, P^2)^{\mathbf{C}(m_1, m_2)}$ is defined as follows: For every $i \in \mathbf{I}((n_1, n_2))$, if $pr_{idx}(i) \neq \perp$, then the local state $t(pr_{idx}(i))$ is defined as $s(i)$.

Given the functions pr_{idx} and pr_{st} , we define a function pr_{path} that maps a run $\pi = s_1 s_2 \dots$ of $(P^1, P^2)^{\mathbf{C}(n_1, n_2)}$ to a sequence of states of $(P^1, P^2)^{\mathbf{C}(m_1, m_2)}$. Note that it is part of the framework to prove that the latter sequence is actually a run. To this end, we partition π into (finitely or infinitely many) paths Π_1, Π_2, \dots with the following properties for every $k \geq 1$.

- The last state of Π_k and the first state of Π_{k+1} form a transition of a process whose index i satisfies $i \in \mathbf{I}((n_1, n_2))$ and $pr_{idx}(i) \neq \perp$.
- Every pair of successive states in Π_k forms a transition of a process whose index i satisfies $i \in \mathbf{I}((n_1, n_2))$ and $pr_{idx}(i) = \perp$.

Let $s_{j(1)} s_{j(2)} \dots$ be the sequence of the first states of Π_1, Π_2, \dots . Then $pr_{path}(\pi)$ is defined as follows.

- If the sequence $\Pi_1 \Pi_2 \dots$ is infinite, then $pr_{path}(\pi) = pr_{st}(s_{j(1)}) pr_{st}(s_{j(2)}) \dots$
- If the sequence $\Pi_1 \Pi_2 \dots$ has length k , then $pr_{path}(\pi) = pr_{st}(s_{j(1)}) pr_{st}(s_{j(2)}) \dots pr_{st}(s_{j(k)}) (pr_{st}(s_{j(k)}))^{\omega}$.

Given a path π , $pr_{path}(\pi)$ is a sequence, not necessarily a run. As pr_{st} and pr_{path} remove processes, the sequence $pr_{path}(\pi)$ is not immediately a path of $(P^1, P^2)^{\mathbf{C}(m_1, m_2)}$. The partitioning of $pr_{path}(\pi)$ implies that $pr_{path}(\pi)$ is *stuttering equivalent* to π ; a state s_i of $pr_{path}(\pi)$ forms its own partition that is equivalent to partition Π_i . Similarly to [Baier and Katoen, 2008, Theorem 7.92], one can prove the following.

Observation 6.18 Let $(P^1, P^2)^{\mathbf{C}(n_1, n_2)}$ and $(P^1, P^2)^{\mathbf{C}(m_1, m_2)}$ be two system instances of a guarded protocol (P^1, P^2) such that $n_2 \geq m_2$ and $n_1 \geq m_1$. Let pr_{idx} be an index projection from $\mathbf{C}(n_1, n_2)$ to $\mathbf{C}(m_1, m_2)$ and $c \in V(\mathbf{C}(n_1, n_2))$ a process index.

Consider an $\text{LTL} \setminus \text{X}$ -formula φ over atomic propositions $\mathcal{Q}^{\text{type}(c)} \times \{c\}$ and an $\text{LTL} \setminus \text{X}$ -formula φ' that is the result of substitution of c with $pr_{idx}(c)$. For each run π of $(P^1, P^2)^{\mathbf{C}(n_1, n_2)}$, the following holds:

$$(P^1, P^2)^{\mathbf{C}(n_1, n_2)}, \pi \models \varphi \text{ if and only if } (P^1, P^2)^{\mathbf{C}(m_1, m_2)}, pr_{path}(\pi) \models \varphi'.$$

Of course, we are interested only in an index projection pr_{idx} that actually generates runs, that is, it turns every run π of $(P^1, P^2)^{\mathbf{C}(n_1, n_2)}$ into a run $pr_{path}(\pi)$ of $(P^1, P^2)^{\mathbf{C}(m_1, m_2)}$. We call such a projection a *proper index projection*.

84 6. GUARDED PROTOCOLS

An index projection removes processes, and thus it cannot increase the number of processes in a local state. This intuition is formulated in the following proposition, which is easy to prove.

Proposition 6.19 *Let $(P^1, P^2)^{\mathbf{C}(n_1, n_2)}$ be a system instance with the set of global states S and pr_{idx} an index projection. For every global state $s \in S$ and every local state $q \in Q^1 \cup Q^2$, it holds that $\kappa(s, q) \geq \kappa(pr_{st}(s), q)$.*

Symmetry argument. Similarly to parameterized token rings (cf. [Emerson and Namjoshi \[1995\]](#)), it was noticed by [Emerson and Kahlon \[2000\]](#) that model checking of a guarded protocol with a fixed number of processes against a formula from $1\text{-ELTL}\backslash X(P^\ell)$ or $1\text{-LTL}\backslash X(P^\ell)$ can be reduced to model checking against an $ELTL\backslash X$ formula over the states of a fixed process:

Observation 6.20

For a system instance $(P^1, P^2)^{\mathbf{C}(m_1, m_2)}$ of a guarded protocol, process type $\ell \in \{1, 2\}$, index c of a process of type ℓ , and an indexed $\{A, E, X\}$ -free path formula $\varphi(i)$, the following holds.

1. $(P^1, P^2)^{\mathbf{C}(m_1, m_2)} \models \forall i : \text{type}(i) = \ell. E\varphi(i)$ if and only if $(P^1, P^2)^{\mathbf{C}(m_1, m_2)} \models E\varphi(c)$.
2. $(P^1, P^2)^{\mathbf{C}(m_1, m_2)} \models \forall i : \text{type}(i) = \ell. A\varphi(i)$ if and only if $(P^1, P^2)^{\mathbf{C}(m_1, m_2)} \not\models E\neg\varphi(c)$.

When having guarded protocols with one controller C and many user processes U , Observation 6.20(2) leads to the following corollary.

Corollary 6.21

For a system instance $(C, U)^{\mathbf{C}(1, m_2)}$ of a guarded protocol (C, U) , and an $\{A, E, X\}$ -free path formula φ over the states of controller C , the following holds: $(C, U)^{\mathbf{C}(1, m_2)} \models A\varphi$ if and only if $(C, U)^{\mathbf{C}(1, m_2)} \not\models E\neg\varphi$.

6.4.2 PROOF SCHEMAS

Based on counting function, path reduction, and symmetry argument, we can define the two proof schemas that are used to prove cutoff results for disjunctive and init-conjunctive guards.

Bounding schema

- i. Estimate the cutoff size (c_1, c_2) based on the number of local states in P^1 and P^2 and the types of guards (disjunctive, init-conjunctive). Typically, one gives an arithmetic expression for c_1 and c_2 over $|Q^1|$ and $|Q^2|$, respectively. Fix system size $(n_1, n_2) \geq (c_1, c_2)$ and an arbitrary run $\pi = s_1 s_2 \dots$ of $(P^1, P^2)^{\mathbf{C}(n_1, n_2)}$, which we use in the rest of the proof. In case of disjunctive guards, we require an intermediate step and π actually is a representative run with a special structure.
- ii. Based on point (i), construct an index projection pr_{idx} from $\mathbf{C}(n_1, n_2)$ to $\mathbf{C}(c_1, c_2)$. This leads to the mapping pr_{path} , as discussed in the path reduction.

- iii. Show that the guards of π are not locked in $pr_{path}(\pi)$. From this follows that the pairs of consecutive states in $pr_{path}(\pi)$ form a transition, and thus that the sequence $pr_{path}(\pi)$ is a path of $(P^1, P^2)^{\mathbf{C}(c_1, c_2)}$. If π is a deadlocked run, then prove that $pr_{path}(\pi)$ is also a deadlocked run.

This actually constitutes the core of the proof. One shows that if a guard evaluates to true in a state s of $(P^1, P^2)^{\mathbf{C}(n_1, n_2)}$, then it is also true in state $pr_{st}(s)$ of $(P^1, P^2)^{\mathbf{C}(c_1, c_2)}$. Typically, if one underestimates (c_1, c_2) at point (i), then the proof breaks at this point.

- iv. Apply Observation 6.18 to conclude that for every index k with $pr_{idx}(k) \neq \perp$ it holds that $(P^1, P^2)^{\mathbf{C}(n_1, n_2)} \models E\varphi(k)$ implies $(P^1, P^2)^{\mathbf{C}(c_1, c_2)} \models E\varphi(pr_{idx}(k))$.

Monotonicity schema

- i. Fix system sizes (m_1, m_2) and $(m_1 + d_1, m_2 + d_2)$ with $d_1 + d_2 = 1$ and $d_1, d_2 \in \{0, 1\}$. Pick an arbitrary run $\pi = t_1 t_2 \dots$ of $(P^1, P^2)^{\mathbf{C}(m_1, m_2)}$. The goal of the next steps is to construct a corresponding run σ of system instance $(P^1, P^2)^{\mathbf{C}(m_1 + d_1, m_2 + d_2)}$.
- ii. Construct a sequence $\sigma = s_1 s_2 \dots$ of global states of $(P^1, P^2)^{\mathbf{C}(m_1 + d_1, m_2 + d_2)}$ by mapping every pair of states (t_i, t_{i+1}) of π on a sequence of states in $(P^1, P^2)^{\mathbf{C}(m_1 + d_1, m_2 + d_2)}$. Typically, the pair of states is replaced by one or two transitions that mimic (t_i, t_{i+1}) .
- iii. Show that the guards of the transitions in σ evaluate to true. Thus, the pairs of consecutive states in σ form a transition, and the sequence σ is a run of $(P^1, P^2)^{\mathbf{C}(m_1 + d_1, m_2 + d_2)}$. If π is a deadlocked run, then prove that σ is also a deadlocked run.
- iv. Construct an index mapping pr_{idx} from $\mathbf{C}(m_1 + d_1, m_2 + d_2)$ to $\mathbf{C}(m_1, m_2)$ as follows: $pr_{idx}(m_1 + d_1 + m_2 + d_2) = \perp$; for all $i \leq m_1 + m_2$, $pr_{idx}(i) = i$. Conclude that $pr_{path}(\sigma) = \pi$.
- v. From $pr_{path}(\sigma) = \pi$, conclude that for every index $k \leq m_1 + m_2$ it holds that $(P^1, P^2)^{\mathbf{C}(m_1, m_2)} \models E\varphi(pr_{idx}(k))$ implies $(P^1, P^2)^{\mathbf{C}(m_1 + d_1, m_2 + d_2)} \models E\varphi(k)$.

In the following sections, we instantiate the bounding and monotonicity proof schemas for init-conjunctive and disjunctive guards. Figures 6.11, 6.12, 6.13, 6.14, and 6.15 illustrate these proofs.

Cutoff argument. Assume that the bounding schema works for a cutoff size (c_1, c_2) . By applying inductively the monotonicity schema and Observation 6.20, we conclude that the following holds for every $n_1 \geq c_1$ and $n_2 \geq c_2$ and every formula ψ from 1-LTL\X and 1-ELTL\X:

$$(P^1, P^2)^{\mathbf{C}(n_1, n_2)} \models \psi \text{ if and only if } (P^1, P^2)^{\mathbf{C}(c_1, c_2)} \models \psi.$$

86 6. GUARDED PROTOCOLS

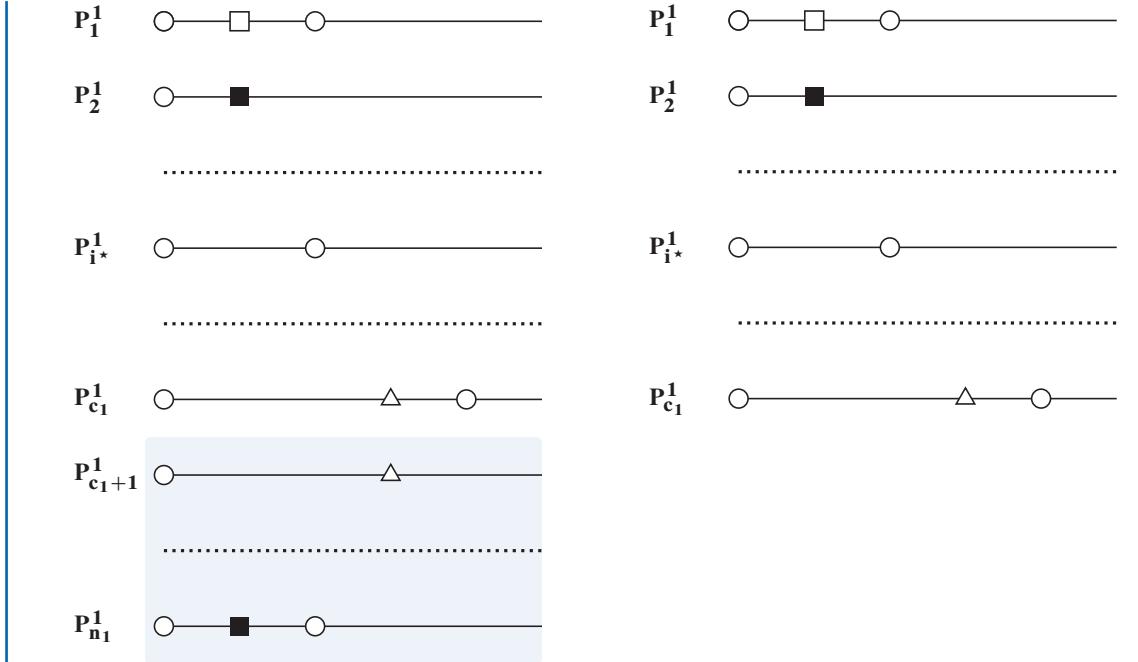


Figure 6.11: Illustration of the bounding schema for the init-conjunctive guards. For simplicity, we left out the processes instantiated from the template P^2 . Local states of the processes are depicted with \circ , \square , \blacksquare , and \triangle . The processes in the shaded area are hidden by the index projection function pr_{idx} , and thus only the processes $P^1_1, \dots, P^1_{c_1}$ are left after the bounding schema has been applied. The process $P^1_{i^*}$ makes infinitely many transitions, unless the original execution is deadlocked. (If the process $P^1_{i^*}$ has an index above c_1 , we first apply the symmetry argument and swap it with an arbitrary process having an index below c_1 , e.g., process P^1_2 .)

6.4.3 INIT-CONJUNCTIVE GUARDS

Here we instantiate the bounding and monotonicity schemas to arrive at the following theorem for init-conjunctive guarded protocols [Emerson and Kahlon, 2000, Theorem 7]).

Theorem 6.22

For number $\ell \in \{1, 2\}$ and a specification class \mathcal{F} from $1\text{-LTL}\backslash X(P^\ell)$, $1\text{-ELTL}\backslash X(P^\ell)$, or $LTL\backslash X(C)$, the problem $PMCP(\mathcal{P}_{\text{init-conj}}[2], C, \mathcal{F})$ is decidable.

Moreover, for every init-conjunctive guarded protocol (P^1, P^2) , there exists a cutoff of size (c_1, c_2) , that is for every formula $\varphi \in \mathcal{F}$, the following two statements are equivalent.

1. For every clique $C(n_1, n_2)$ it holds $(P^1, P^2)^{C(n_1, n_2)} \models \varphi$.
2. For every clique $C(m_1, m_2)$ with $m_1 \leq c_1$ and $m_2 \leq c_2$, it holds $(P^1, P^2)^{C(m_1, m_2)} \models \varphi$.

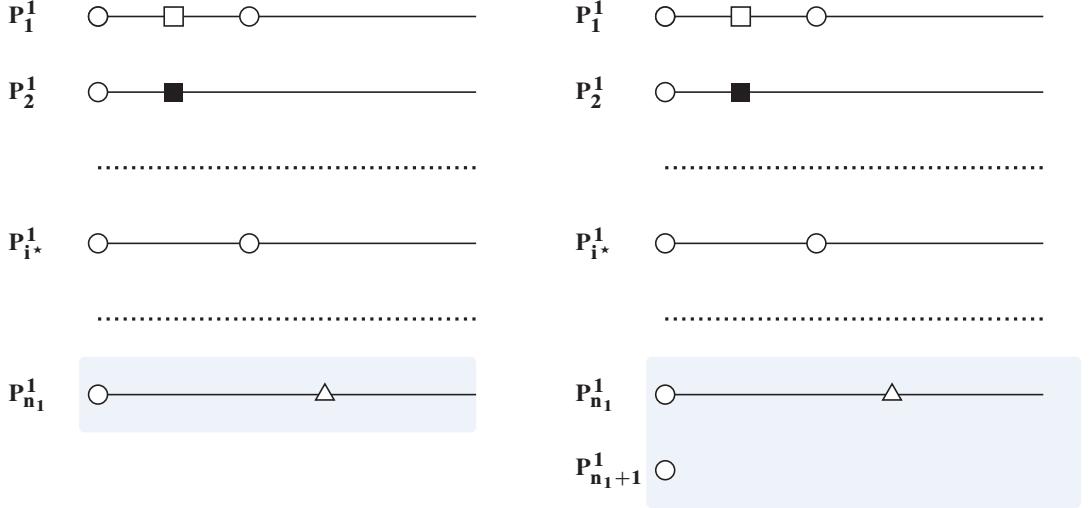


Figure 6.12: Illustration of the monononicity schema for init-conjunctive guards. For simplicity, we left out the processes instantiated from the template P^2 . Local states of the processes are depicted with \circlearrowleft , \square , \blacksquare , and \triangle . The additional process $P_{n_1+1}^1$ never leaves the initial state \circlearrowleft .

The cutoff size is defined by $c_\ell = 2|Q^\ell| + 1$ and $c_{3-\ell} = 2|Q^{3-\ell}|$.

For the rest of the section, we fix a guarded protocol (P^1, P^2) with init-conjunctive guards and a $\{A, E, X\}$ -free formula $\varphi(1)$ over the states of process 1. As in Theorem 6.22, we fix cutoff size to be $c_1 = 2|Q^1| + 1$ and $c_2 = 2|Q^2|$.

For bounding schema we consider two cases: (1) a run is free of deadlocks; and (2) a run has a deadlock.

(1) *Bounding schema for deadlock-free runs:*

- i. Let $\pi = s_1 s_2 \dots$ be a deadlock-free run of $(P^1, P^2)^{\mathbf{C}(n_1, n_2)}$. This implies that there is a process that performs infinitely many steps in π . Let i^* be the index of such a process.
- ii. Construct an index projection pr_{idx} from $\mathbf{C}(n_1, n_2)$ to $\mathbf{C}(c_1, c_2)$ that has the following properties: (a) $pr_{idx}(1) = 1$; (b) $pr_{idx}(i^*) \neq \perp$. Note carefully that by the definition of index projection, exactly $c_1 + c_2$ indices of $\mathbf{C}(n_1, n_2)$ are mapped on $\mathbf{C}(c_1, c_2)$.
- iii. We have to show that for every $i \geq 1$, the guard of the transition from s_i to s_{i+1} is evaluated to true in $pr_{st}(s_i)$. Consider the transition from s_i to s_{i+1} made by process p . Let its guard g_i be $[\forall \text{ other } p] q_1 \vee \dots \vee q_k$. As $s_i \models_p g_i$, all processes are in a state from $\{q_1, \dots, q_k\}$, and for every $q \in Q^1 \cup Q^2 \setminus \{q_1, \dots, q_k\}$ it holds that $\kappa(s_i, q) = 0$. Hence, from Proposition 6.19, it immediately follows that $\kappa(pr_{st}(s_i), q) = 0$. From the latter we conclude that $pr_{st}(s_i) \models_{pr_{idx}(p)} g_i$. Thus, $pr_{path}(\pi)$ constitutes a run of $(P^1, P^2)^{\mathbf{C}(c_1, c_2)}$.

88 6. GUARDED PROTOCOLS

- iv. Apply Observation 6.18 to conclude that for every index k with $pr_{idx}(k) \neq \perp$ it holds that $(P^1, P^2)^{\mathbf{C}(n_1, n_2)} \models E\varphi(k)$ implies $(P^1, P^2)^{\mathbf{C}(c_1, c_2)} \models E\varphi(pr_{idx}(k))$.

(2) *Bounding schema for deadlocked runs:*

- i. Let $\pi = s_1 s_2 \dots s_d (s_d)^\omega$ be a deadlocked run of $(P^1, P^2)^{\mathbf{C}(n_1, n_2)}$.
- ii. We construct an index projection pr_{idx} from $\mathbf{C}(n_1, n_2)$ to $\mathbf{C}(c_1, c_2)$ with the following properties:
 - (a) $pr_{idx}(1) = 1$; and
 - (b) for every $q \in Q^1 \cup Q^2$, $\kappa(pr_{st}(s_d), q) \geq \min(\kappa(s_d, q), 2)$.

Such a mapping exists, as $c_1 + c_2 = 2 \cdot (|Q^1| + |Q^2|) + 1$.

- iii. As in the deadlock-free case, for every $i \geq 1$ and every $q \in Q^1 \cup Q^2$, $\kappa(s_i, q) \geq \kappa(pr_{path}(s_i), q)$. It follows that for every $i : 1 \leq i < d$ and every process index $p : 1 \leq p \leq n_1 + n_2$, $s_i \models_p g_i \Rightarrow s_i \models_{pr_{idx}(p)} g_i$.

It remains to prove that $pr_{st}(s_d)$ is a deadlock state, that is, for every guard g and some process index p , it holds $s_d \not\models_p g \Rightarrow pr_{st}(s_d) \not\models_{pr_{idx}(p)} g$. Let us fix g to be $[\forall \text{ other } p] q_1 \vee \dots \vee q_k$ and a process index p . As $s_d, p \not\models g$, there is a state $q \in Q^1 \cup Q^2 \setminus \{q_1, \dots, q_k\}$ with the following properties:

- (a) if $s_d(p) = q$, then $\kappa(s_d, q) \geq 2$; and
- (b) if $s_d(p) \neq q$, then $\kappa(s_d, q) \geq 1$.

From point (ii) we have $\kappa(pr_{st}(s_d), q) \geq \min(\kappa(s_d, q), 2)$. Hence, in case (a), we have $\kappa(pr_{st}(s_d), q) \geq 2$. In case (b), we have $\kappa(pr_{st}(s_d), q) \geq 1$. In both cases we conclude that $pr_{st}(s_d) \not\models_{pr_{idx}(p)} g$. Thus, $pr_{st}(s_d)$ is a deadlock state, as required.

- iv. Apply Observation 6.18 to conclude that for every index k with $pr_{idx}(k) \neq \perp$ it holds that $(P^1, P^2)^{\mathbf{C}(n_1, n_2)} \models E\varphi(k)$ implies $(P^1, P^2)^{\mathbf{C}(c_1, c_2)} \models E\varphi(pr_{idx}(k))$.

Monotonicity schema:

- i. Fix system sizes (m_1, m_2) and $(m_1 + d_1, m_2 + d_2)$ for $\ell \in \{1, 2\}$ and $d_1 = \ell \cdot (2 - \ell)$ and $d_2 = \ell \cdot (\ell - 1)$. Pick an arbitrary run $\pi = t_1 t_2 \dots$ of $(P^1, P^2)^{\mathbf{C}(m_1, m_2)}$.
- ii. Map every pair (t_i, t_{i+1}) on a pair (s_i, s_{i+1}) , where for all $j : 1 \leq j \leq m_1 + m_2$, $s_i(j) = t_i(j)$, $s_{i+1}(j) = t_{i+1}(j)$ and $s_i(m_1 + m_2 + d_1 + d_2) = s_{i+1}(m_1 + m_2 + d_1 + d_2) = init^\ell$.
- iii. We have to show that for every guard g , every process index p , and every $i \geq 1$, if $t_i \models_p g$, then $s_i \models_{pr_{idx}(p)} g$. This is immediate, as, by the definition, every init-conjunctive guard contains $init^\ell$, which is the only state the added process resides in.

- iv. Construct an index mapping pr_{idx} from $\mathbf{C}(m_1 + d_1, m_2 + d_2)$ to $\mathbf{C}(m_1, m_2)$ as follows: $pr_{idx}(m_1 + d_1 + m_2 + d_2) = \perp$; for all $i \leq m_1 + m_2$, $pr_{idx}(i) = i$. Conclude that $pr_{path}(\sigma) = \pi$.
- v. From $pr_{path}(\sigma) = \pi$, conclude that for every index $k \leq m_1 + m_2$ it holds that $(P^1, P^2)^{\mathbf{C}(m_1, m_2)} \models E\varphi(pr_{idx}(k))$ implies $(P^1, P^2)^{\mathbf{C}(m_1 + d_1, m_2 + d_2)} \models E\varphi(k)$.

Figure 6.11 illustrates the bounding schema, and Figure 6.12 illustrates the monotonicity schema.

By applying the bounding and monotonicity schemas, we finish the proof of Theorem 6.22.

Further Results. As one can see from the proof of Theorem 6.22, if the system instances are deadlock-free, there is a trivial cutoff.

Theorem 6.23 [Emerson and Kahlon, 2000]. Consider a guarded protocol (P^1, \dots, P^d) with conjunctive guards and a formula φ from $1\text{-LTL}\backslash X(P^\ell)$, $1\text{-ELTL}\backslash X(P^\ell)$, or $LTL\backslash X(C)$ for some $\ell \in [d]$. If every instance $(P^1, \dots, P^d)^{\mathbf{C}(n)}$ does not have deadlocked runs, then there is a cutoff system of size $(1, \dots, 1, 2, 1, \dots, 1)$, i.e., the following two statements are equivalent.

1. For each $(n_1, \dots, n_d) \geq (1, \dots, 1)$, it holds $(P^1, \dots, P^d)^{\mathbf{C}(n_1, \dots, n_d)} \models \varphi$.
2. The cutoff instance of size (c_1, \dots, c_d) , where $c_\ell = 2$ and $\forall k \neq \ell. c_k = 1$, satisfies the formula: $(P^1, \dots, P^d)^{(c_1, \dots, c_d)} \models \varphi$.

A similar argument works for properties that require only finite paths, e.g., reachability and safety.

Theorem 6.24 [Emerson and Kahlon, 2000]. Consider a d -ary conjunctive system template (P^1, \dots, P^d) and a formula φ from $1\text{-LTL}\backslash X(P^\ell)$, $1\text{-ELTL}\backslash X(P^\ell)$, or $LTL\backslash X(C)$ for some $\ell \in [d]$. If φ can be verified on finite computations, then the system of size $(1, \dots, 1)$ is a cutoff: for each $(n_1, \dots, n_d) \geq (1, \dots, 1)$, it holds $(P^1, \dots, P^d)^{\mathbf{C}(n_1, \dots, n_d)} \models \varphi$ if and only if $(P^1, \dots, P^d)^{\mathbf{C}(1, \dots, 1)} \models \varphi$.

6.4.4 DISJUNCTIVE GUARDS

Emerson and Kahlon [2000] obtained the following cutoff result for systems with disjunctive guards.

Theorem 6.25 For number $\ell \in \{1, 2\}$ and a specification class \mathcal{F} from $1\text{-LTL}\backslash X(P^\ell)$, $1\text{-ELTL}\backslash X(P^\ell)$, or $LTL\backslash X(C)$, the problem $PMCP(\mathcal{P}_{disj}[2], \mathbf{C}, \mathcal{F})$ is decidable.

Moreover, for every disjunctive guarded protocol (P^1, P^2) , there exists a cutoff of size (c_1, c_2) , that is for every formula $\varphi \in \mathcal{F}$, the following two statements are equivalent.

1. For every clique $\mathbf{C}(n_1, n_2)$ it holds $(P^1, P^2)^{\mathbf{C}(n_1, n_2)} \models \varphi$.

90 6. GUARDED PROTOCOLS

2. For every clique $\mathbf{C}(m_1, m_2)$ with $m_1 \leq c_1$ and $m_2 \leq c_2$, it holds $\overline{P}^{\mathbf{C}(m_1, m_2)} \models \varphi$.

The cutoff size is defined by $c_\ell = |Q^\ell| + 2$ and $c_{3-\ell} = |Q^{3-\ell}| + 1$.

To prove Theorem 6.25, we instantiate the bounding and monotonicity schemas. The most important part of the proof, however, is established in the “Unlocking” lemma that follows after the technical definition of a local computation of a process in a run of a system instance. In the following definition, we project a run on one process and remove stuttering introduced by the other processes.

Definition 6.26 Let $\pi = s_1, \dots, s_k$ be a run of a guarded system instance $(P^1, P^2)^{\mathbf{C}(n_1, n_2)}$ and p be the index of a process of type $\ell \in \{1, 2\}$. Further, let $i(1), \dots, i(m)$ be a sequence of numbers in $[k]$ with the following properties:

- $i(1) = 1$;
- for all $j : 1 \leq j < m$, $i(j) < i(j+1)$ and $(s_{i(j)}(p), s_{i(j+1)}(p)) \in \delta^\ell$; and
- for all $j : 1 \leq j < m$, for all $h : 1 \leq h < i(j)$, it holds $(s_{i(j)+h}(p), s_{i(j)+h+1}(p)) \notin \delta^\ell$.

We call the sequence $s_{i(1)}(p), \dots, s_{i(m)}(p)$ a *local computation* of p in s_1, \dots, s_k and denote it as $(s_1, \dots, s_k) \downarrow p$.

Definition 6.27 Let π be run $s_1 s_2 \dots$ of a guarded instance $(P^1, P^2)^{\mathbf{C}(n_1, n_2)}$. We define the set of local reachable states as $Reach(\pi) = \{q \mid \exists i \geq 1, \exists p : 1 \leq p \leq n_1 + n_2, s_i(p) = q\}$.

In the following lemma, we assign a unique process to each local state from $Reach(\pi)$. That local state is called a *goal state*.

Lemma 6.28 Unlocking Let π be a run $s_1 s_2 \dots$ of a guarded system instance $(P^1, P^2)^{\mathbf{C}(n_1, n_2)}$ that has only disjunctive guards. There exists a run π' of $(P^1, P^2)^{\mathbf{C}(n_1 + |Q^1|, n_2 + |Q^2|)}$ organized as follows: π' has a prefix, where each of the at most $|Q^1| + |Q^2|$ processes runs into its unique goal state from $Reach(\pi)$ and remains there; the prefix is followed by transitions of $n_1 + n_2$ processes as in π . The run π' is stuttering equivalent to π w.r.t. to $n_1 + n_2$ processes.

Proof. We construct π' and then exploit that the transitions have only disjunctive guards to show that π' is a run. To this end, for each $q \in Q^1 \cup Q^2$, we define the distance of q from the initial state in π as $dist(q) = \min\{i \mid \exists p : 1 \leq p \leq n_1 + n_2, s_i(p) = q\}$, if $q \in Reach(\pi)$, and ∞ otherwise.

Let q_1, \dots, q_k be the sequence of all local states from $Reach(\pi)$, ordered with respect to $dist$. Due to interleaving semantics, only one process takes a step at a time. Hence, for every $q, q' \in Reach(\pi)$, $q \neq q'$, it holds $dist(q) \neq dist(q')$. Due to this fact, we can define a function $first : Reach(\pi) \rightarrow [n_1 + n_2]$ that maps a reachable state q to the index of the process that performed the transition $dist(q)$. As all processes are initially either in $init^1$ or $init^2$, we define $first(init^1) = n_1$

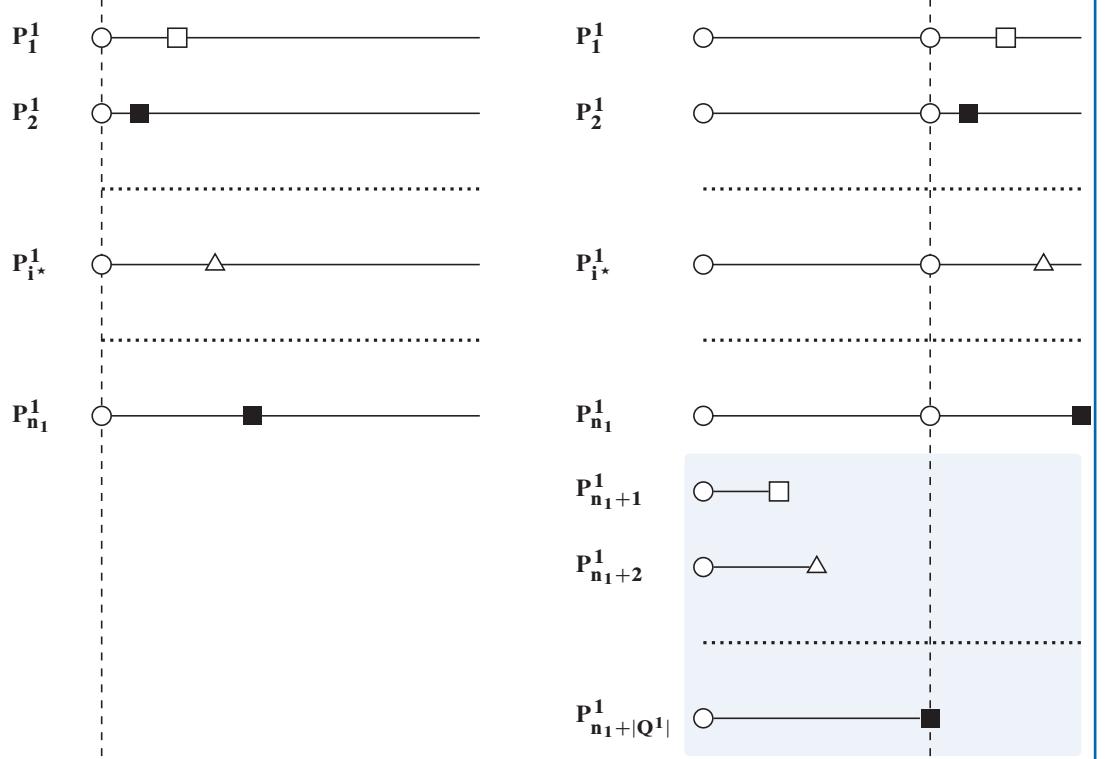


Figure 6.13: Illustration of the unlocking lemma (Lemma 6.28) for the disjunctive guards. For simplicity, we left out the processes instantiated from the template P^2 . Local states of the processes are depicted with \circlearrowleft , \square , \blacksquare , and \triangle . The shaded area on the right highlights the processes added by the unlocking lemma. Each of these processes halts after reaching its designated state.

and $\text{first}(\text{init}^2) = n_2$. Note that first does not have to be an injection, that is, it maps several local states to the index of the same process, if the process is the first one to visit these states.

We construct a set of local computations $\{\rho_i \mid 1 \leq i \leq k\}$ with $\rho_i = (s_1, \dots, s_{\text{dist}(q_i)}) \downarrow \text{first}(q_i)$ for each $i : 1 \leq i \leq k$. Now we define a set of paths $\{y_1^i, \dots, y_{|\rho_i|}^i \mid 1 \leq i \leq k\}$ that belong to the system instance $(P^1, P^2)^{\mathbf{C}(n_1 + |Q^1|, n_2 + |Q^2|)}$. For each $i : 1 \leq i \leq k$, each $j : 1 \leq j \leq |\rho_i|$, and each process index $p : 1 \leq p \leq n_1 + |Q^1| + n_2 + |Q^2|$ we define a local state $y_j^i(p)$ of process p as follows:

$$y_j^i(p) = \begin{cases} \rho_i[j] & \text{if } p = n_1 + n_2 + i, 0 \leq i \leq |\text{Reach}(\pi)| \\ \rho_i[|\rho_i'|] & \text{if } \text{dist}(q_i') < \text{dist}(q_i), p = n_1 + n_2 + i', 0 \leq i' \leq |\text{Reach}(\pi)| \\ \text{init}^{\text{type}(p)} & \text{otherwise.} \end{cases}$$

92 6. GUARDED PROTOCOLS

Then we extend the global states s_1, s_2, \dots with the halting states of the new $|Q^1| + |Q^2|$ processes, that is, for each $j \geq 1$:

$$s'_j(p) = \begin{cases} s_j(p) & \text{if } p \leq n_1 + n_2 \\ \rho_i[|\rho_i|] & \text{if } p = n_1 + n_2 + i, 0 \leq i \leq |\text{Reach}(\pi)| \\ \text{init}^{\text{type}(p)} & \text{otherwise.} \end{cases}$$

Finally, we construct a sequence π' as a $(y_1^1, \dots, y_{|\rho_1|}^1), \dots, (y_1^k, \dots, y_{|\rho_k|}^k), s'_1, s'_2, \dots$. Now we show that π' is actually a run.

First, we show that the suffix $s'_1, s'_2 \dots$ forms a path of $(P^1, P^2)^{\mathbf{C}(n_1 + |Q^1|, n_2 + |Q^2|)}$. As the transitions have only disjunctive guards, and the states of the processes with indices below $n_1 + n_2$ remain as in π , the processes with the indices above $n_1 + n_2$ do not disable the guards of the transitions made by the processes with the indices below $n_1 + n_2$. On the contrary, all the guards that are required to fire the transitions of π become enabled after executing the sequence $(y_1^1, \dots, y_{|\rho_1|}^1), \dots, (y_1^k, \dots, y_{|\rho_k|}^k)$. Formally, let $g \equiv [\exists \text{ other } j] q_1 \vee \dots \vee q_m$ be a disjunctive guard, s_i a state on π , and $p \leq n_1 + n_2$ a process index such that $s_i \models_p g$. Then there is a process index $p' \leq n_1 + n_2$ and a local state $q \in \{q_1, \dots, q_m\}$ with $s_i(p') = q$. As $q \in \text{Reach}(\pi)$, there is a process index $p'' > n_1 + n_2$ with $y_{n_k}^k(p'') = q$, moreover, for all $i \geq 1$, $s'_i(p'') = q$. Hence, for all $i \geq 1$, $s'_i \models_p g$.

Second, we prove that the prefix $(y_1^1, \dots, y_{|\rho_1|}^1), \dots, (y_1^k, \dots, y_{|\rho_k|}^k)$ is a run of $(P^1, P^2)^{\mathbf{C}(n_1 + |Q^1|, n_2 + |Q^2|)}$. As we define each ρ_i as a local computation, the sequent states are related with δ^1 or δ^2 . The guards in $y_1^i, \dots, y_{|\rho_i|}^i$ are enabled, as we ordered q_1, \dots, q_k by dist , and π witnessed the shortest local computations ρ_1, \dots, ρ_k , leading to the local states in $\text{Reach}(\pi)$.

The run $(y_1^1, \dots, y_{|\rho_1|}^1), \dots, (y_1^k, \dots, y_{|\rho_k|}^k)$ does not change the local states of the processes with the indices below $n_1 + n_2$. Thus, π' is stuttering equivalent to π w.r.t. the indices below $n_1 + n_2$.

Hence, π' is the required run of $(P^1, P^2)^{\mathbf{C}(n_1 + |Q^1|, n_2 + |Q^2|)}$. □

Figure 6.13 illustrates application of the unlocking lemma.

We are now in the position to prove Theorem 6.25 according to the proof schema.
Bounding schema:

- i. Let $\pi = s_1 s_2 \dots$ be a run of $(P^1, P^2)^{\mathbf{C}(n_1, n_2)}$. If π is a deadlock-free run, then let i^* be the index of a process that fires infinitely many transitions in π , otherwise i^* is an arbitrary index different from 1. Construct a stuttering equivalent run π' of $(P^1, P^2)^{\mathbf{C}(n_1 + |Q^1|, n_2 + |Q^2|)}$ as stated in Lemma 6.28.
- ii. Construct an index projection pr_{idx} from $\mathbf{C}(n_1 + |Q^1|, n_2 + |Q^2|)$ to $\mathbf{C}(c_1, c_2)$ as follows:

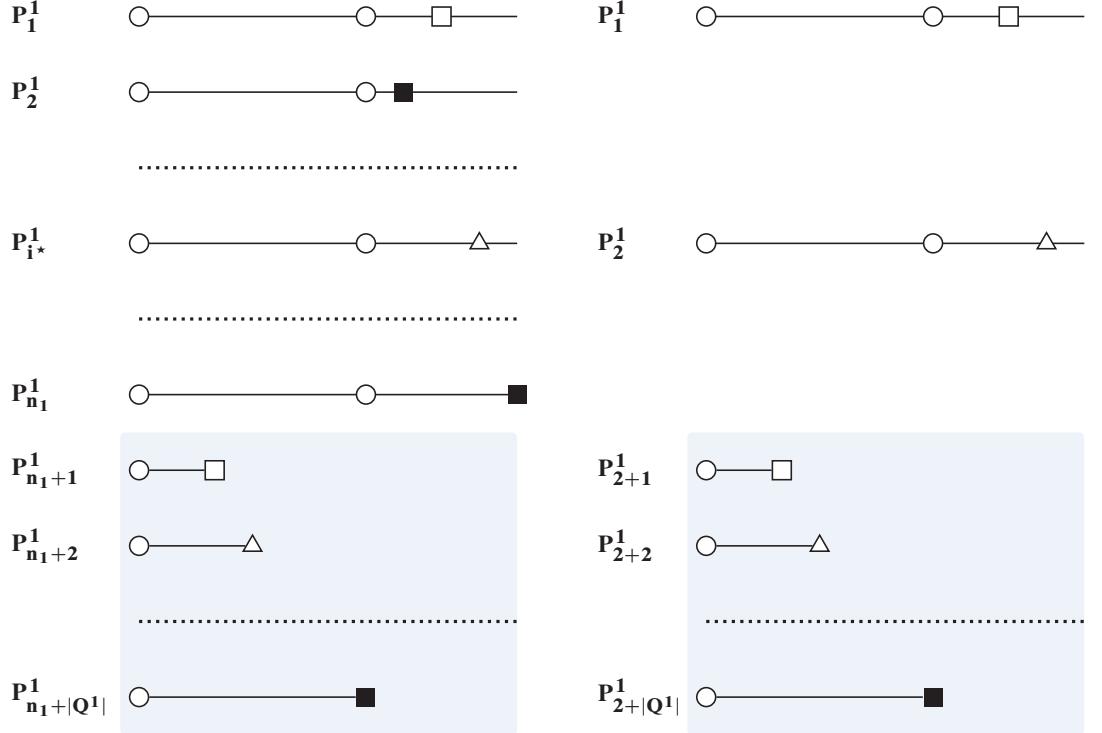


Figure 6.14: Illustration of the bounding schema applied together with the unlocking lemma (Lemma 6.28). For simplicity, we left out the processes instantiated from the template P^2 . Local states of the processes are depicted with \circlearrowleft , \square , \blacksquare , and \triangle . Process P_1^1 is kept as is, because its behavior is observed by the specification. Process $P_{i^*}^1$ is mapped to process P_2^1 , as it makes infinitely many transitions (if the execution is deadlock-free). Every process $P_{n_1+i}^1$ is mapped to process P_{2+i}^1 for $1 \leq i \leq |Q^1|$.

$$pr_idx(i) = \begin{cases} 1 & \text{if } i = 1 \\ 2 & \text{if } i = i^* \\ 2 + (i - (n_1 + n_2)) & \text{if } n_1 + n_2 < i \leq n_1 + n_2 + |Q_1| + |Q_2| \\ \perp & \text{otherwise.} \end{cases}$$

- iii. As we keep the processes with the indices above $n_1 + n_2$, and they fire transitions before the processes with the indices below $n_1 + n_2$, their guards remain enabled. The guards of the transitions by the processes with the indices 1 and i^* are enabled by the local states of the

94 6. GUARDED PROTOCOLS

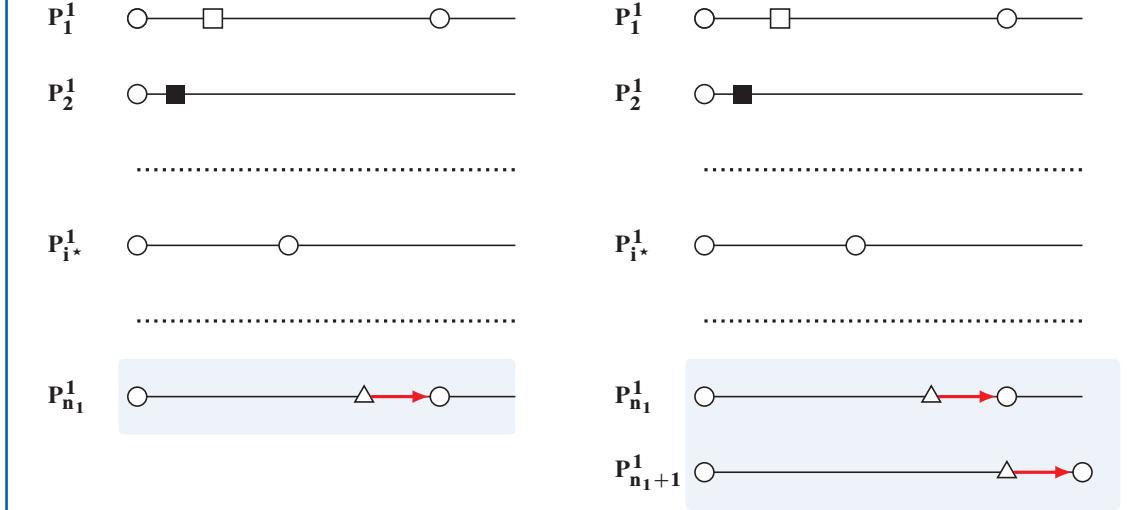


Figure 6.15: Illustration of the monononicity schema for disjunctive guards. For simplicity, we left out the processes instantiated from the template P^2 . Local states of the processes are depicted with \circ , \square , \blacksquare , and \triangle . The additional process $P_{n_1+1}^1$ mimics every transition made by the process $P_{n_1}^1$.

processes with the indices above $n_1 + n_2$, as constructed in Lemma 6.28. Hence, $pr_{path}(\pi')$ is a run of $(P^1, P^2)^{\mathbf{C}(c_1, c_2)}$.

- iv. From stuttering equivalence of π and π' and Observation 6.18, we conclude that for every index k with $pr_{idx}(k) \neq \perp$ it holds that $(P^1, P^2)^{\mathbf{C}(n_1, n_2)} \models E\varphi(k)$ implies $(P^1, P^2)^{\mathbf{C}(c_1, c_2)} \models E\varphi(pr_{idx}(k))$.

Figure 6.14 illustrates application of the bounding schema together with the unlocking lemma.

Remark 6.29 In the proof of Lemma 6.28, we move specific processes to cover all states in *Reach*. Thus, if the run π has a deadlock state s_d , it might happen that s'_d is not a deadlock state. To repair this, similar to Unlocking lemma, one can introduce a suffix that moves the processes with indices above $n_1 + n_2$ out of $Reach(\pi) \setminus \{q \mid \exists p. s_d(p) = q\}$ states.

Monotonicity schema:

- i. Fix system sizes (m_1, m_2) and $(m_1 + d_1, m_2 + d_2)$ with $d_1, d_2 \in \{0, 1\}$ and $d_1 + d_2 = 1$. Pick an arbitrary run $\pi = t_1 t_2 \dots$ of $(P^1, P^2)^{\mathbf{C}(m_1, m_2)}$.
- ii. By N we abbreviate the index $m_1 + m_2$. Then $N + 1 = m_1 + d_1 + m_2 + d_2$. Map every pair t_i, t_{i+1} as follows.

Case 1. If (t_i, t_{i+1}) is a transition by process with index N , then we map it on a triple s_i, s'_i, s_{i+1} , where $\forall p : 1 \leq p < m_1 + m_2$. $s_i(p) = t_i(p)$, $s'_i(p) = t_i(p)$, $s_{i+1}(p) = t_{i+1}(p)$ and $s_i(N) = t_i(N)$, $s'_i(N) = s_{i+1}(N) = t_{i+1}(N)$ and $s_i(N+1) = s'_i(N+1) = t_i(N)$, $s_{i+1}(N+1) = t_{i+1}(N+1)$. Informally, the process with index $N+1$ copies the transition of the process with index N .

Case 2. If (t_i, t_{i+1}) is a transition by process $p \neq N$, then we map it on a pair s_i, s_{i+1} , where $\forall p : 1 \leq p \geq m_1 + m_2$. $s_i(p) = t_i(p)$, $s_{i+1}(p) = t_{i+1}(p)$ and $s_i(N+1) = t_i(N)$, $s_{i+1}(N+1) = t_{i+1}(N)$.

- iii. In case (ii.1), we have to show that for every guard g and every process index $p < m_1 + m_2$, if $t_i \models_p g$ holds then $s_i \models_p g$ and $s'_i \models_p g$ also hold. Let g be $[\exists \text{ other } q_1 \vee \dots \vee q_k]$. From $t_i \models_p g$ we conclude that there is a process $p' \leq m_1 + m_2$ and $p' \neq p$ with $t_i(p') \in \{q_1, \dots, q_k\}$. From (ii), $s_i(p') = s'_i(p') = t_i(p')$. Hence, $s_i \models_p g$ and $s'_i \models_p g$. In case (ii.2), we immediately obtain that guards are preserved in every state s_i .
- iv. Apply Observation 6.18 to conclude that for every index k with $pr_{idx}(k) \neq \perp$ it holds that $(P^1, P^2)^{\mathbf{C}(m_1, m_2)} \models E\varphi(pr_{idx}(k))$ implies $(P^1, P^2)^{\mathbf{C}(m_1+d_1, m_2+d_2)} \models E\varphi(k)$.

Figure 6.15 illustrates application of the monotonicity schema.

The unlocking lemma, the bounding schema, and the monotonicity schema applied together finish the proof of Theorem 6.25.

Further Results. Emerson and Kahlon [2000] also extended the cutoff result on two index specifications:

Theorem 6.30 *Let ℓ and ℓ' be process types and \mathcal{F} be one of $2\text{-LTL} \setminus X(P^\ell, P^{\ell'})$ and $2\text{-ELTL} \setminus X(P^\ell, P^{\ell'})$. The problem $PMCP(\mathcal{P}_{\text{disj}}, \mathbf{C}, \mathcal{F})$ is decidable. Moreover, there is a cutoff of size (c_1, \dots, c_d) , where $c_\ell = |Q^\ell| + 1$ and $c_{\ell'} = |Q^{\ell'}| + 1$ and for all other $k \in [d] \setminus \{\ell, \ell'\}$, $c_k = |Q^k|$.*

The estimates of cutoffs in Theorems 6.25 and 6.30 require model checking of quite large systems containing at least $|Q^\ell|$ processes of each type ℓ . Emerson and Kahlon [2000] gave an efficient solution for universal-path-quantified formulas. We do not give details here.

Emerson and Kahlon [2003b] noted that the PMC problem for protocols with disjunctive guards is decidable for regular and ω -regular properties over actions.

Theorem 6.31 *$PMCP(\mathcal{P}_{\text{disj}}, \mathbf{C}, \mathcal{F})$ is decidable for $\mathcal{F} = Reg(A)$ and $\mathcal{F} = \omega Reg(A)$.*

Proof Sketch. The intuition is that the behavior of all the systems instantiated from a disjunctive system template (P^1, \dots, P^d) can be encoded as a Petri net \mathcal{P} . The language of \mathcal{P} is checked for intersection with the specification, i.e., the language of either a Büchi automaton, or a finite automaton over actions. This problem is known to be decidable [Esparza and Nielsen, 1994, Theorem 8].

96 6. GUARDED PROTOCOLS

The constructed Petri net \mathcal{P} has one place per local state from $Q^1 \cup \dots \cup Q^d$; the number of tokens in a place q encodes the number of processes in local state q . If a process makes a transition from one local state to another, then the number of tokens in the source place and the target places are decremented and incremented, respectively. A disjunctive guard $[\exists \text{ other } i] q_1 \vee \dots \vee q_k$ is then encoded as k transitions—one per local state q_i —fetching a token from q_i and then returning it back to q_i . \square

6.5 DISJUNCTIVE GUARDS VS. RENDEZVOUS

In this section we consider the relation between systems with disjunctive guards and pairwise rendezvous. We review results by [Emerson and Kahlon \[2003b\]](#) who showed that the systems with disjunctive guards and the systems with pairwise rendezvous are reducible to each other in the sense that the reduction preserves properties from $\text{Reg}(\mathbf{A})$ and $\omega\text{Reg}(\mathbf{A})$ (formally defined below).

For $\text{LTL} \setminus \text{X}(\mathbf{C})$ properties, we show that the given reductions (designed for $\text{Reg}(\mathbf{A})$ and $\omega\text{Reg}(\mathbf{A})$) work only in one direction: the provided reduction from disjunctive guards to pairwise rendezvous preserves $\text{LTL} \setminus \text{X}(\mathbf{C})$ properties. However, for the other direction, the provided reduction from pairwise rendezvous to disjunctive guards is not conservative with respect to $\text{LTL} \setminus \text{X}(\mathbf{C})$.

First, we introduce the notions of ξ -reducibility ([Emerson and Kahlon \[2003b\]](#) denoted it via \prec_ϕ) and a special case of stuttering trace equivalence. Let us fix AP and two LTSs $M_i = (Q_i, Q_i^0, \Sigma_i, \delta_i, \lambda_i)$ for $i = 1, 2$. Furthermore, let $\xi : \Sigma_1 \rightarrow \Sigma_2^+$ be an action mapping; as common we extend ξ to a language $L \subseteq \Sigma_1^* \cup \Sigma_1^\omega$ as $\xi(L) = \{\xi(w) : w \in L\}$.

We say that M_1 is ξ -reducible to M_2 if for every regular language $L \subseteq \Sigma_1^*$ and every ω -regular language $L_\omega \subseteq \Sigma_1^\omega$ the following holds.

- Properties of *finite* action-labeled runs of M_1 and M_2 are indistinguishable by L and $\xi(L)$, i.e., $\mathcal{L}(M_1) \cap L = \emptyset$ iff $\mathcal{L}(M_2) \cap \xi(L) = \emptyset$.
- Properties of *infinite* action-labeled runs of M_1 and M_2 are indistinguishable by L_ω and $\xi(L_\omega)$, i.e., $\xi(\mathcal{L}_\omega(M_1)) \cap L_\omega = \emptyset$ iff $\mathcal{L}_\omega(M_2) \cap \xi(L_\omega) = \emptyset$.

Thus, if M_1 is ξ -reducible to M_2 , then one can reduce model checking of an action-based property of M_1 , which is using one kind of synchronization primitives, to model checking of the corresponding property of M_2 , which is using another set of synchronization primitives. Note carefully that ξ -reducibility does not require the languages of M_1 and M_2 to be equivalent (up to ξ), hence, the language of M_2 may contain action runs that are not in $\xi(\Sigma_1)$.

Furthermore, for a (finite or infinite state-labeled) path $\pi : i \mapsto q_i$ we call the sequence $tr(\pi) : i \mapsto \lambda(q_i)$ the *trace* of π . Let $stutter_k$ be the function defined as $stutter_k(w_1 w_2 \dots) = (w_1)^k (w_2)^k \dots$ for any word $w_1 w_2 \dots \in AP^* \cup AP^\omega$, i.e., every letter of the word is repeated k times. We say that M_1 is k -equivalent to M_2 , if the traces of the systems coincide up to proposition stuttering, i.e., $\{stutter_k(tr(\pi)) : \pi \text{ is a run of } M_1\} = \{tr(\pi') : \pi' \text{ is a run of } M_2\}$.

One can see that k -equivalence implies stutter trace equivalence. Thus, using the well-known fact about preservation of $\text{LTL} \setminus X$ properties by stutter trace equivalent systems (cf. Baier and Katoen [2008, Theorem 7.92]), we obtain: *For k -equivalent LTSs M_1 and M_2 and for $\text{LTL} \setminus X$ -formula ψ , the following holds: $M_1 \models \psi$ iff $M_2 \models \psi$.*

Now we prove an interesting fact: Every system with disjunctive guards is ξ -reducible and 2-equivalent to a system with pairwise rendezvous (shown by Emerson and Kahlon [2003b, Section 5]):

Theorem 6.32 *For every guarded system template (P^1, \dots, P^d) there exists a system template $(\tilde{P}^1, \dots, \tilde{P}^d)$ using pairwise rendezvous such that for every clique $\mathbf{C}(n)$, the system instance $(P^1, \dots, P^d)^{\mathbf{C}(n)}$ is ξ -reducible and 2-equivalent to the system instance $(\tilde{P}^1, \dots, \tilde{P}^d)^{\mathbf{C}(n)}$.*

Proof. Let every process template P^ℓ be $(Q^\ell, Q_0^\ell, \Sigma^\ell, \delta^\ell, \lambda^\ell)$. Following Emerson and Kahlon [2003b], we assume that the set of actions Σ_{pr} is a disjoint union $\delta^1 \cup \dots \cup \delta^d$, and every transition is labeled with a unique action from Σ_{pr} . For every $\ell \in [d]$, we construct a process template $\tilde{P}^\ell = (\tilde{Q}^\ell, \tilde{Q}_0^\ell, \tilde{\Sigma}^\ell, \tilde{\delta}^\ell, \tilde{\lambda}^\ell)$ as follows.

- Set of states \tilde{Q}^ℓ is the disjoint union of the original states Q^ℓ and of a set of auxiliary states defined by $\{\text{syn}(t) : t \in \delta^\ell\} \cup \{\text{cosyn}(c, t) : c \in Q^1 \cup \dots \cup Q^d \text{ and } t \in \delta^\ell\}$.
- Initial states stay the same: $\tilde{Q}_0^\ell = Q_0^\ell$.
- State labels $\tilde{\delta}^\ell$ are assigned as follows: Original states $q \in Q^\ell$ are labeled with $\{q\}$; Synchronization states $\text{syn}((q, a, q')) \in Q^\ell$ are labeled with their predecessor $\{q\}$; Co-synchronization states $\text{cosyn}(c, t) \in Q^\ell$ are labeled with their predecessor $\{c\}$.
- $\tilde{\Sigma}^\ell = \{\text{req}(t)! : t \in \delta^\ell\} \cup \{\text{ack}(t)? : t \in \delta^\ell\} \cup \{\text{ack}(t)! : t \in \bigcup_{k \in [d]} \delta^k\} \cup \{\text{req}(t)? : t \in \bigcup_{k \in [d]} \delta^k\}$.
- For every transition $t = (q, a, q')$ from δ^ℓ one adds to $\tilde{\delta}^\ell$ two transitions $(q, \text{req}(t)!, \text{syn}(t))$ and $(\text{syn}(t), \text{ack}(t)?, q')$. Furthermore, for every proposition c_i of the disjunctive guard $\text{gd}(t) = [\exists \text{ other } j] c_1 \vee \dots \vee c_k$, one adds two transitions $(c_i, \text{req}(t)?, \text{cosyn}(t))$ and $(\text{cosyn}(t), \text{ack}(t)!, c_i)$ to the corresponding transition relation $\tilde{\delta}^\ell$ of the process template containing the state c_i .

It is easy to see that every transition $t = (q, a, q')$ of P^ℓ is translated to a sequence of transitions $q \xrightarrow{\text{req}(t)!} \text{syn}(t) \xrightarrow{\text{ack}(t)?} q'$ made by a process of type ℓ ; this sequence is synchronized with a sequence $c_i \xrightarrow{\text{req}(t)?} \text{cosyn}(c_i, t) \xrightarrow{\text{ack}(t)!} c_i$ executed by a process residing in state c_i . The intermediate state $\text{cosyn}(c_i, t)$ stores its predecessor c_i to return back to it later.

By construction, the intermediate states $\text{syn}(t)$ and $\text{cosyn}(c_i, t)$ preserve the labels of their predecessors. Thus, every letter in a trace of $(P^1, \dots, P^d)^{\mathbf{C}(n)}$ is doubled in $(\tilde{P}^1, \dots, \tilde{P}^d)^{\mathbf{C}(n)}$.

98 6. GUARDED PROTOCOLS

Define a mapping $\xi(t)$ as $req(t)! \cdot ack(t)?$. Now it is easy to see that every system instance $(P^1, \dots, P^d)^{\mathbf{C}(n)}$ is ξ -reducible and 2-equivalent to the system instance $(\tilde{P}^1, \dots, \tilde{P}^d)^{\mathbf{C}(n)}$. \square

Emerson and Kahlon [2003b, Section 5] showed a reduction of systems with pairwise rendezvous to systems with disjunctive guards, while preserving ξ -reducibility.

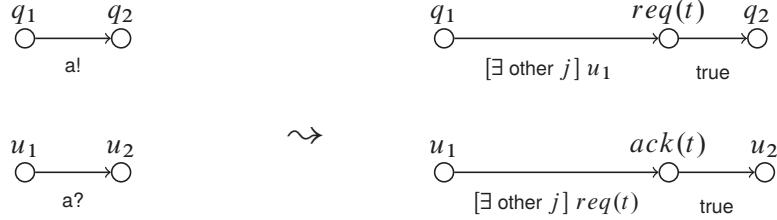


Figure 6.16: Reducing *with caveat* pairwise rendezvous (left) to transitions with disjunctive guards (right).

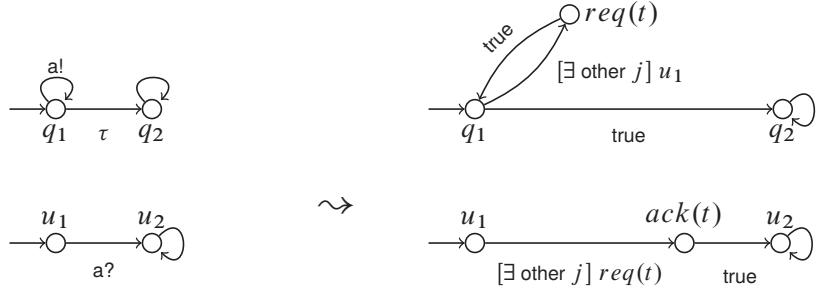


Figure 6.17: An example showing that the reduction on Figure 6.16 does not preserve formulas from $LTL \setminus X(C)$.

Theorem 6.33 For each system template (P^1, \dots, P^d) with pairwise rendezvous there is a guarded protocol $(\tilde{P}^1, \dots, \tilde{P}^d)$ with disjunctive guards having the following property: For every clique $\mathbf{C}(n)$, system instance $(P^1, \dots, P^d)^{\mathbf{C}(n)}$ is ξ -reducible to system instance $(\tilde{P}^1, \dots, \tilde{P}^d)^{\mathbf{C}(n)}$.

Proof idea. The proof similar to the proof of Theorem 6.32. Figure 6.16 shows how a pair of synchronized transitions $t = (q_1, a!, q_2)$ and $(u_1, a?, u_2)$ is translated to four transitions with disjunctive guards. The intermediate states $req(t)$ and $ack(t)$ keep the labels of their predecessors, i.e., of q_1 and u_1 . Given process templates, such a translation is done for every pair of transitions carrying synchronization actions of the same type.

Let b , c , d , and e be the actions assigned to the transitions $q_1 \rightarrow req(t)$, $req(t) \rightarrow q_2$, $u_1 \rightarrow ack(t)$, $ack(t) \rightarrow u_2$, respectively. Then the mapping ξ of $a!$ is defined as $\xi(a!) = bcde$.

Thus, a finite (or Büchi) automaton checking a specification has to recognize the sequence $bcd e$ instead of letter $a!$, which corresponds to handshake of two processes issuing actions $a!$ and $a?$. \square

In contrast to action-based properties, state-based properties are not preserved by the reduction in the proof of Theorem 6.33.

Observation 6.34 The reduction provided in the proof of Theorem 6.33 does not preserve $LTL \setminus X(C)$ properties.

Proof. Figure 6.17 shows a system template (C, U) with rendezvous (left-hand side) and its reduction to a guarded protocol (C', U') with disjunctive guards (righthand-side). Fix an arbitrary $n \in \mathbb{N}$. For the system with rendezvous, it holds that $(C, U)^{C(1,n)} \models F q_2$, as n processes of type U , allow C to fire $(q_1, a!, q_1)$ finitely many times, that is, every user process moves from u_1 to u_2 once. In the system with disjunctive guards, C' can fire $(q_1, [\exists \text{ other } j] u_1, req(t))$ infinitely often, without a single process of type U' moving. Hence, $(C', U')^{C(1,n)} \not\models F q_2$. \square

6.6 VARIATIONS ON THE MODEL: GUARDS AND SYNCHRONIZATION PRIMITIVES

The protocols we considered so far had guarded internal transitions only. Emerson and Kahlon [2003a,b,c] investigated the cases of guarded systems with broadcast actions, pairwise rendezvous, and asynchronous rendezvous. These combined systems allow one to model, e.g., cache coherence protocols.

Most of the results for the combined models are obtained using reduction techniques by Emerson and Kahlon [2003b]. The results are summarized in Figure 6.1. Here we briefly discuss the ideas behind the results. Most of the proofs are based on the undecidability results for (rendezvous and broadcast) systems without guards, and the reduction provided in Section 6.5.

Disjunctive guards and rendezvous $\mathcal{P}_{\text{disj\&rdvz}}$. Decidability of the PMCP for $\mathcal{P}_{\text{disj\&rdvz}}$ (disjunctive guards and pairwise rendezvous) depends on the specification class. Decidability for $\text{Reg}(A)$ is given by Emerson and Kahlon [2003b]; the result relies on backward reachability in well-structured systems, which is decidable [Abdulla et al., 1996]. Further, as the systems with disjunctive guards are reducible to systems with pairwise rendezvous (see Theorem 6.32) one can re-use the decidability proof for rendezvous by German and Sistla [1992] (cf. Section 5). Namely, the PMCP is also decidable for the classes of $LTL \setminus X(C)$, $LTL(C)$, $1-LTL \setminus X(P^\ell)$, and $\omega\text{Reg}(A)$ properties.

Disjunctive guards and broadcast $\mathcal{P}_{\text{disj\&bcast}}$. In the case of $\mathcal{P}_{\text{disj\&bcast}}$ (disjunctive guards and broadcasts), none of the specification classes mentioned in this section are decidable except $\text{Reg}(A)$. As disjunctive guards are reducible to pairwise rendezvous, and systems with pairwise rendezvous and broadcasts are reducible to systems with broadcasts (see Section 5 and Esparza et al. [1999]), we can reduce systems in $\mathcal{P}_{\text{disj\&bcast}}$ to systems with broadcast. As Esparza et al. [1999] showed,

100 6. GUARDED PROTOCOLS

the PMCP is undecidable for all classes except $\text{Reg}(A)$; in the case of $\text{Reg}(A)$, the construction of a coverability graph by [Emerson and Namjoshi \[1998\]](#) applies.

Disjunctive guards and asynchronous rendezvous $\mathcal{P}_{\text{disj}\&\text{ardvz}}$. The PMCP for $\mathcal{P}_{\text{disj}\&\text{ardvz}}$ (disjunctive guards and asynchronous rendezvous) is also undecidable, except for the case of $\text{Reg}(A)$, which is similar to the result for $\mathcal{P}_{\text{disj}\&\text{rdvz}}$. As shown by [Emerson and Kahlon \[2003b\]](#), the PMCP of $\text{LTL} \setminus \text{X}(\mathbf{C})$ properties is undecidable for systems with both pairwise rendezvous and asynchronous rendezvous. Due to this and reducibility of disjunctive guards to pairwise rendezvous, it immediately follows that the PMCP is undecidable for all specification classes that include $\text{LTL} \setminus \text{X}(\mathbf{C})$.

Conjunctive guards and synchronization. Disjunctive guards are reducible to pairwise rendezvous (see [Theorem 6.32](#)). Thus, PMCP for $\mathcal{P}_{\text{conj+disj}}$ (in other words, PMCP for boolean guards, $\mathcal{P}_{\text{bool}}$) can be reduced to PMCP for $\mathcal{P}_{\text{conj+rdvz}}$ (conjunctive guards and pairwise rendezvous). As PMCP is undecidable for boolean guards and every class of specifications (see [Theorem 6.13](#)), PMCP for $\mathcal{P}_{\text{conj+rdvz}}$ and every class of specifications is undecidable as well.

Furthermore, as shown by [Emerson and Kahlon \[2003b\]](#), pairwise rendezvous is reducible to asynchronous rendezvous. As we have just shown that PMCP for $\mathcal{P}_{\text{conj+rdvz}}$ is undecidable for every class of specifications, it follows that PMCP for $\mathcal{P}_{\text{conj+ardvz}}$ (conjunctive guards and asynchronous rendezvous) is also undecidable for every class of specifications. Similarly, PMCP is undecidable for $\mathcal{P}_{\text{conj}\&\text{bcast}}$ (conjunctive guards and broadcasts). Again, this is due to reducibility of disjunctive guards to pairwise rendezvous and of pairwise rendezvous to broadcast [[Esparza et al., 1999](#)]. Together with the undecidability results for boolean guards, this shows undecidability for all considered classes of specifications.

6.7 DISCUSSION

In this chapter, we introduced the results on guarded protocols in the unified framework. This allows us to present different computational models from the literature as instances of our definitions. We re-structure the proofs of the cutoff theorems for disjunctive and init-conjunctive guards, which highlight central arguments of these important results. [Emerson and Kahlon \[2003b\]](#) claimed in their discussions that there is a decidability result for conjunctive guards. However, the reference they give is based on init-conjunctive guards [[Emerson and Kahlon, 2000](#)]. As the proofs use the fact that an init-conjunctive guard cannot be disabled by a process at the initial state, it is not straightforward to generalize these proofs to conjunctive guards. Hence, to the best of our understanding, decidability for system with *conjunctive guards* and specifications from $1\text{-LTL} \setminus \text{X}(\mathbf{P}^\ell)$, $1\text{-ELTL} \setminus \text{X}(\mathbf{P}^\ell)$, or $\text{LTL} \setminus \text{X}(\mathbf{C})$ are open problems.

We clarified that the reduction of disjunctive guards to rendezvous presented in [Emerson and Kahlon \[2003b\]](#) is limited to action-based specifications. In particular, we showed that disjunctive guards are not powerful enough to capture rendezvous with respect to $\text{LTL}(\mathbf{C})$ properties.

Guarded protocols are interesting not only from a theoretical point of view, but are also useful in modeling cache coherence protocols—an important concept for hardware designs. The

following papers discuss applications of guarded protocols in modeling and verification of cache coherence protocols. There are two notable papers by [Emerson and Kahlon \[2003a,c\]](#) showing how guarded protocols with broadcasts can be applied to modeling of cache coherence protocols. In the mentioned papers, the authors consider restrictions of guarded systems with one process template U .

- The conjunctive guards have the form $[\forall \text{ other } i] init$.
- The disjunctive guards have the form $[\exists \text{ other } i] \neg init$.
- Apart from guards, processes use broadcasts.
- The transitions of process templates have various structural restrictions, e.g., from every state the process is able to go to $init$.

[Emerson and Kahlon \[2003a,c\]](#) modeled eight cache coherence protocols with the framework of guarded systems. As the PMCP for guarded systems with broadcast is undecidable, the authors had to develop special abstraction and cutoff techniques for the protocols with the restrictions mentioned above. As the results seem to be considerably tailored to the special structure of cache coherence protocols, we do not give details in this survey. An interested reader finds a detailed exposition in the original papers.

Some lower bounds are known for disjunctively guarded systems: PMCP is undecidable for 1-index $\text{CTL}^*\backslash X$ specifications [Aminof et al. \[2014b\]](#); for systems with or without a controller the complexity of the PMCP is PSPACE-complete (the upper bound is from [Emerson and Kahlon \[2000\]](#) and the lower bound is from [Aminof et al. \[2014b\]](#)); for systems with a controller the program complexity (i.e., the formula is fixed and not part of the input) is coNP-complete [Aminof et al. \[2014b\]](#), whereas for systems without a controller the program complexity is in PTIME (the upper bound is from [Emerson and Kahlon \[2000\]](#) and the lower bound is from [Aminof et al. \[2014b\]](#)).

From a theoretical point of view, we observe from the discussions in this chapter that there are the following open problems.

Problem 6.35 Is the PMCP with conjunctive guards and the following specifications decidable: $1\text{-LTL}\backslash X(P^\ell)$, $2\text{-LTL}\backslash X(P^\ell, P^{\ell'})$, $\text{LTL}(C)$? Is there a cutoff, similar to Theorem 6.22?

Problem 6.36 Is the PMCP with disjunctive guards and the following specifications decidable: prenex $\text{ILTL}\backslash X$, prenex $\text{ICTL}^*\backslash X$, $\text{ILTL}\backslash X$?

CHAPTER 7

Ad Hoc Networks

In ad hoc networks, processes communicate via broadcasts. Recall from Section 2.2.1 that in transitions synchronized by broadcast, all immediate neighbors of a sender simultaneously take a transition with the sender, and the sender is not blocked if there are no recipients that are ready to synchronize. This chapter summarizes the results by [Delzanno et al. \[2010, 2011, 2012b\]](#) and [Abdulla et al. \[2013a\]](#).

The literature studies ad hoc networks arranged in different classes of connectivity graphs that try to model typical topologies met in practice (see Section 7.4), and also networks in which broadcast messages can be lost and thus not received by some of the sender's neighbors. In case of (non-lossy) ad hoc networks (AHN), parameterized model checking problems, even for safety properties, are undecidable for systems with general connectivity graphs, but become decidable for certain classes of graphs. In case of lossy ad hoc networks (LAHN), the additional non-determinism of whether or not a message is lost for a receiving process makes all parameterized model checking problems considered in this chapter decidable. However, it is well known that important problems cannot be solved in the presence of lossy links, e.g., the two generals paradox [[Gray, 1978](#)]. A related model is that of *mobile* ad hoc networks, in which the structure of the communication graph can change during execution. It has been shown by [Delzanno et al. \[2012b\]](#) that such networks are equivalent to a certain class of lossy ad hoc networks. Generally, the models discussed in this section, though inspired by ad hoc networks (cf. [Karl \[2005\]](#)), have not yet found their practical applications, so that in the next section we provide a toy example to help us to illustrate the theoretical results.

7.1 RUNNING EXAMPLE

Consider the process template in Figure 7.1. Processes start in the initial state *init*. Processes that are in state *u* will be called “user” processes, processes in *l* are “leader” processes, processes in *hub* are “hub” processes, and processes in *err* are “error” processes. Figure 7.2 gives two connectivity graphs with reachable global states of the system in AHNs or LAHNs, respectively. For AHNs composed of processes running this protocol, the following holds:

- (i) in AHNs arranged on any undirected graph, state *err* is unreachable;
- (ii) in AHNs arranged in cliques, state *hub* is unreachable; and
- (iii) in AHNs arranged on any undirected graph, any process adjacent to a leader is not a leader (and will eventually be either a hub or a user process).

Consider the first item. Let process p_{err} be one of the processes that reach state err first (hypothetically, there may be several processes that reach err at the same time). Let us assume that p_{err} reaches err via transition $l' \xrightarrow{req??} err$. Also, wlog. assume p_{err} is the first among the error processes that reaches l' . Just before getting into l' the process sends $req!!$ which makes it impossible for any adjacent process to send $req!!$. The last fact implies that p_{err} cannot reach err from l' . Contradiction. In a similar way, we can derive a contradiction for the case of reaching err via transition $l \xrightarrow{req??} req$. Later in this chapter, in Example 7.20 we prove this formally by applying an algorithm that can answer reachability questions of this kind.

The items (i)–(iii) do not hold for LAHNs, namely: on cliques, states err and hub are reachable, and leader processes can be adjacent. Later in this chapter, Example 7.22 proves item (i) adapted to LAHNs on cliques formally. An example of the protocol execution that falsifies (iii) for LAHNs is in Figure 7.2.

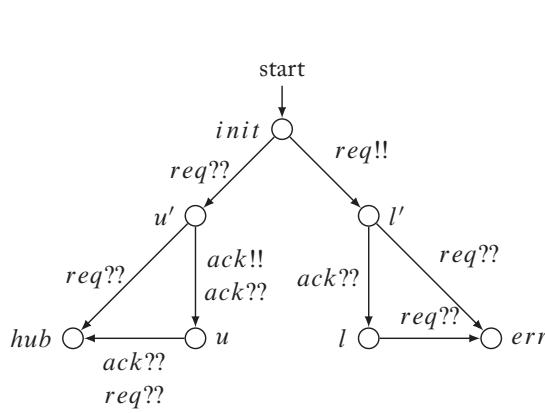


Figure 7.1: Process template for the running example.

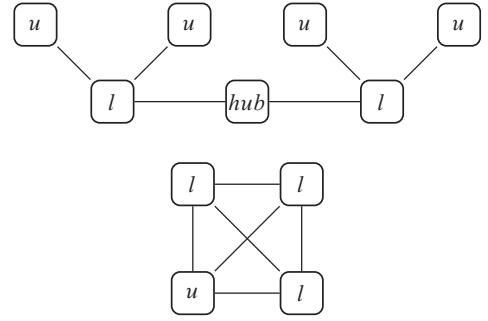


Figure 7.2: On top: reachable global state after executing the protocol in an AHN. Processes are labeled with their local state. On the bottom: reachable global state after executing the protocol in a LAHN.

7.2 SYSTEM MODEL

Process and system template. The only communication primitive allowed is a broadcast as defined in Section 2.2.1, i.e., the synchronization constraint is $\text{card} = \mathbb{N}_0$. In the literature, only systems with a single process template P are considered, i.e., $d = 1$. Denote by $\mathcal{P}_{\text{bcast}}$ the set of all process templates that communicate via broadcasts.

1-ary parameterized connectivity graph \mathbf{G} . In this chapter, we consider 1-ary parameterized connectivity graphs \mathbf{G} as a *set* of 1-colored connectivity graphs, rather than a sequence as defined in Section 2.2. This is simply a matter of convenience, as the order of graphs will not matter.

AHN system instance P^G . The definition of AHN system instance P^G follows that of Section 2.2.

LAHN system instance P_{lossy}^G . To define LAHN system instance we introduce a *lossy broadcast transition* by dropping the (MAX) requirement from the definition in Section 2.2.1. Recall that the (MAX) requirement forces all processes that can take a “receive” broadcast transition to take it. Dropping the (MAX) requirement allows a process to “ignore” it: for every process that is a recipient of the initiator and can take a transition labeled with some $b??$, whether it actually takes it is chosen non-deterministically. Informally, this may be viewed as a “message loss.” Denote by P_{lossy}^G a system instance with all broadcast transitions being lossy.

Mobile AHNs. In *mobile ad hoc networks*, the underlying connectivity graph can change during a run of the system. We will not include such changes of the connectivity graph into our system model. However, it has been shown by [Delzanno et al., 2012b, Proposition 1] that mobile ad hoc networks are equivalent to LAHNs on cliques, and therefore the decidability results for LAHNs on cliques also apply to mobile ad hoc networks.

Example 7.1 Figure 7.1 illustrates a process template from $\mathcal{P}_{\text{bcast}}$. Figure 7.2 gives global states that can be reached in an AHN and a LAHN, respectively, after execution of the protocol.

Deadlocks. In this chapter, a run is a finite or infinite maximal path that starts in the initial state, i.e., we handle finite paths directly and neither extend nor ignore deadlocked system runs. Note that the problem of ensuring the absence of deadlocks in parameterized AHNs is undecidable already for AHNs on cliques; this can be proven using a modified construction from Theorem 5.11. In contrast, this problem for LAHNs and AHNs is decidable for parameterized cliques and general graphs, since we can reduce it to the deadlock detection problem in Petri nets, which is decidable [Cheng et al., 1993].

7.3 PMC PROBLEMS FOR AD HOC NETWORKS

For ad hoc networks, three special PMC problems were studied in the literature—**COVER**, **REPEAT**, and **TARGET**.

- **COVER_G**, or *control state reachability*:
input: process template $P = (Q, Q_0, \Sigma, \delta, \lambda) \in \mathcal{P}_{\text{bcast}}$, control states $C \subseteq Q$
output: “Yes” if there exists $G = (V, E) \in \mathbf{G}$ such that system P^G has a run s_0, s_1, \dots (finite or infinite) where for some process index $i \in V$ and some state s_n of the run: $s_n(i) \in C$. That is, there is a system where some process reaches a control state. “No” otherwise.
- **REPEAT_G**, or *repeated control state reachability*:
input: process template $P = (Q, Q_0, \Sigma, \delta, \lambda) \in \mathcal{P}_{\text{bcast}}$, control states $R \subseteq Q$
output: “Yes” if there exists $G = (V, E) \in \mathbf{G}$ such that P^G has an infinite run $\sigma = s_0, s_1, \dots$ where for some process index $i \in V$ and some state s_n of the run: $s_n(i) \in R$. That is, there is a system where some process reaches a control state infinitely often. “No” otherwise.

106 7. AD HOC NETWORKS

where for some process index $i \in V$ the set $\{k \mid s_k(i) \in R\}$ is infinite. That is, there is a system where some process can visit a control state infinitely often. “No” otherwise.

- $\text{TARGET}_{\mathbf{G}}$, or *target reachability*:

input: process template $P = (Q, Q_0, \Sigma, \delta, \lambda) \in \mathcal{P}_{\text{broadcast}}$, control states $T \subseteq Q$

output: “Yes” if there exists $G = (V, E) \in \mathbf{G}$ such that system P^G has a run $\sigma = s_0, s_1, \dots$ (finite or infinite) where for all process indices $i \in V$ $s_n(i) \in T$. That is, there is a system that reaches a global state with all processes being in a control state (the control state may be different for different processes). “No” otherwise.

Under certain conditions (for example, in systems where all runs infinite), these PMC problems can be expressed in indexed temporal logic¹. In particular:

- any instance of $\text{COVER}_{\mathbf{G}}$ can be expressed as a negation of an instance of the PMCP with respect to a 1-indexed safety property;
- any instance of $\text{REPEAT}_{\mathbf{G}}$ —with respect to a universal 1-index persistence property; and
- any instance of $\text{TARGET}_{\mathbf{G}}$ —with respect to a safety property in non-prenex fragment.

The PMC problems for LAHNs are defined similarly, except that P^G is replaced with P_{lossy}^G . [Delzanno et al. \[2012b\]](#) studied PMC problems for LAHN arranged in cliques. We write $P_{\text{lossy}}^{\mathbf{C}}$ for such parameterized systems, where \mathbf{C} is a parameterized clique. To the best of our knowledge, the case of LAHN on other parameterized connectivity graphs was not studied in the literature.

Table 7.1 (on page 128) gives an overview of the results that we discuss in this chapter.

Example 7.2 Now we can relate instances of PMCPs from Section 7.1 with COVER with fixed inputs. Denote the process template from the running example by P_{re} . Then:

- PMC in (i) is equivalent to $\text{COVER}_{\text{All}}$ with $C = \{\text{err}\}$ and $P = P_{re}$, where All is the set of all undirected connected graphs;
- PMC in (ii) is equivalent to $\text{COVER}_{\mathbf{C}}$ with $C = \{\text{hub}\}$ and $P = P_{re}$, where \mathbf{C} is the set of all cliques; and
- PMC (iii) cannot be expressed directly via an instance of COVER , TARGET or REPEAT . But in case of AHNs we can express the PMC in (iii) via an instance of COVER if we modify the process template: add an outgoing broadcast transition from state l , and a corresponding receive transition from l into a special state s . Then, the PMC in (iii) becomes equivalent to $\text{COVER}_{\text{All}}$ with $C = \{s\}$ and $P = P_{re}$.

¹Recall that in this survey we defined the temporal logics on infinite runs only.

7.4 PARAMETERIZED CONNECTIVITY GRAPHS \mathbf{BP}_k , \mathbf{BPC}_k , \mathbf{BD}_k , \mathbf{C} , All

Delzanno et al. [2010, 2011, 2012b] and Abdulla et al. [2013a] studied the PMC problems in ad hoc networks for different classes of parameterized connectivity graphs. The connectivity graphs studied model common topologies met in ad hoc networks. These topologies can be divided into two classes—flat and hierarchical (cf. Karl [2005, Chapter 10]).

In hierarchical topologies, nodes are grouped into clusters, and each cluster has a representative node. All nodes in a cluster communicate directly or via other nodes of the same cluster. Nodes from different clusters do not communicate directly but rather via representative nodes. Below we define bounded path graphs (\mathbf{BP}_k) and bounded path clique graphs (\mathbf{BPC}_k) that can model hierarchical topologies.

In contrast to hierarchical topologies, in flat topologies there are no representative nodes, and every node is equal with respect to communication. We will define bounded diameter graphs (\mathbf{BD}_k) that model flat topologies with a bounded distance between nodes. Cliques (\mathbf{C}) are a special case where the distances is always 1, and can be used to model dense ad hoc networks in which every node is in the transmission range of every other node.

The assumption of undirected connection between nodes does not always hold as the discrepancy between a node's transmission and reception range makes the following possible: a node can send messages to some other node, but cannot receive a message from this node. Connectivity graphs with directed edges can model nodes with such discrepancies.

We use the following basic definitions. A *path in a graph* $G = (V, E)$ is a sequence $v_0e_0v_1e_1 \dots v_n \in (VE)^*V$ such that $(v_i, v_{i+1}) \in e_i$ for $i < n$. It is a *simple path* if no node appears twice. A graph is *connected* if there is a path between any two nodes of the graph. The *distance* between two nodes is the number of edges on the shortest simple path between them. The *length* of a simple path is the number of edges on it, thus the length of $v_0e_0v_1e_1 \dots v_n \in (VE)^*V$ is n . A graph $G = (V, E)$ is *undirected* if for any $(v, w) \in E$ it holds that $(w, v) \in E$.

In the following definitions, let $k \in \mathbb{N}_0$.

All, daAll, C. All is the parameterized connectivity graph consisting of all undirected connected graphs, **daAll** the parameterized connectivity graph of all (directed) acyclic graphs, and **C** is the parameterized connectivity graph of all clique graphs.

BD_k. The *diameter* of a connected undirected graph is the maximal distance between any two nodes of the graph. An undirected connected graph is a *k-bounded diameter graph* if its diameter is less or equal to k . Denote by **BD_k** the parameterized connectivity graph of all k -bounded diameter graphs. Intuitively, the graph diameter equals the maximal number of times a message needs to be broadcast in order to reach all nodes.

BP_k, daBP_k. A graph is a *k*-bounded path graph if all simple paths are of length smaller or equal to *k*. BP_k consists of all undirected connected *k*-bounded path graphs. daBP_k consists of all directed acyclic *k*-bounded path graphs. Note that for any $k \in \mathbb{N}_0$: BP_k \subseteq BD_k.

BPC_k. An undirected connected graph is a *k*-bounded path maximal cliques graph if all simple paths between any of its maximal cliques have lengths smaller or equal to *k*. BPC_k consists of all *k*-bounded path maximal cliques graphs. In terms of expressivity, BPC_k is between BP_k and BD_{2k+1}: for any $k \geq 1$, it holds that BP_k \subset BPC_k \subset BD_{2k+1}.

Example 7.3 Some examples of graphs as defined above.

- A 1-bounded diameter graph is a clique, and BD₁ consists of all cliques, i.e., is equal to C.
- An undirected star is a 2-bounded path graph, the set of all undirected rings with a central node connected to everyone (“wheel”) is in BD₂ but not in BP_k for any *k*.
- Undirected trees of depth *d* are contained in BP_{2d}.
- A directed forest of depth *k* where all edges are directed away from the roots is contained in daBP_k.
- Finally, BPC₀ coincides with the set of all cliques, and stars of cliques are in BPC₂.

7.5 RESULTS FOR (NON-LOSSY) AHNS

The results of Delzanno et al. [2010, 2011, 2012b] for the PMCPs in the classes of graphs defined above are summarized in Figure 7.3.

In the figure the results for the clique column follow from Esparza et al. [1999].

Corollary 7.4 COVER_C is decidable, REPEAT_C and TARGET_C are undecidable for AHNs.

Proof idea. The first item: it is straightforward to reduce COVER_C for AHNs to PMCP(\mathcal{P} , C₀, Reg(A)) for broadcast systems which is decidable by Theorem 5.7. The proof reduces the PMCP problem to the (decidable) coverability problem for WSTSs. The second and third items follow from a straightforward modification of the construction described in Theorem 5.11 that reduces the (undecidable) non-halting problem for 2CMs to PMCP(\mathcal{P} , C₀, ω Reg(A)). \square

7.5.1 UNDECIDABILITY RESULTS FOR (NON-LOSSY) AHNS

This section explains all undecidability results for AHNs from Figure 7.3, namely:

- Theorem 7.6 proves that COVER_{All} is undecidable;
- Theorems 7.8 and 7.9 cover the case of graphs BD_k;

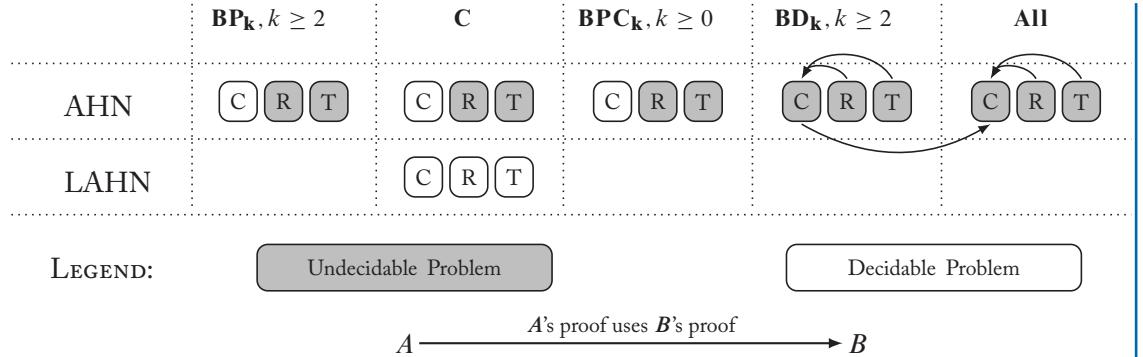


Figure 7.3: Decidability results and reductions for ad hoc networks. The letters c, r, t in the cells stand for COVER, REPEAT, TARGET. The blank cells mean that to the best of our knowledge the problems were not studied in the literature. The results for directed graphs are not shown here—see Theorems 7.12 and 7.19.

- Theorem 7.10 proves that REPEAT and TARGET are undecidable on $\mathbf{BP_k}$ for $k \geq 2$;
- Corollary 7.11 proves that REPEAT and TARGET are undecidable on $\mathbf{BPC_k}$ for $k \geq 0$; and
- Theorem 7.12 proves that COVER is undecidable on directed graphs (not in the figure).

In the literature, undecidability results for ad hoc networks are usually established via reduction from the halting problem for 2CMs to COVER or, equivalently, from the non-halting problem for 2CMs to the PMC problem.

Most of the undecidability proofs build, for a given 2CM, a protocol that has the following two steps:

1. find a sub-graph of a special form, and
2. simulate the 2CM. The simulation assumes the connectivity graph is of the special form.

Building block: RAO protocol. All the protocols used in Delzanno et al. [2010, 2011] to prove undecidability results for All, $\mathbf{BD_k}$, $\mathbf{BP_k}$, $\mathbf{BPC_k}$ as the first step use a variant of the Req-Ack-OK (RAO) protocol (Figure 7.4) that has the following property.

Lemma 7.5 [Delzanno et al., 2010, Proposition 1] Let P be the RAO protocol (Figure 7.4(a)), and consider an AHN system instance P^G . If P^G , starting from the initial state, reaches a system state such that some process $v \in V_G$ is in state s_3 , then:

1. process v is adjacent to exactly one process in state s_3 denoted by w ; and
2. any other process adjacent to v or w is in state err.

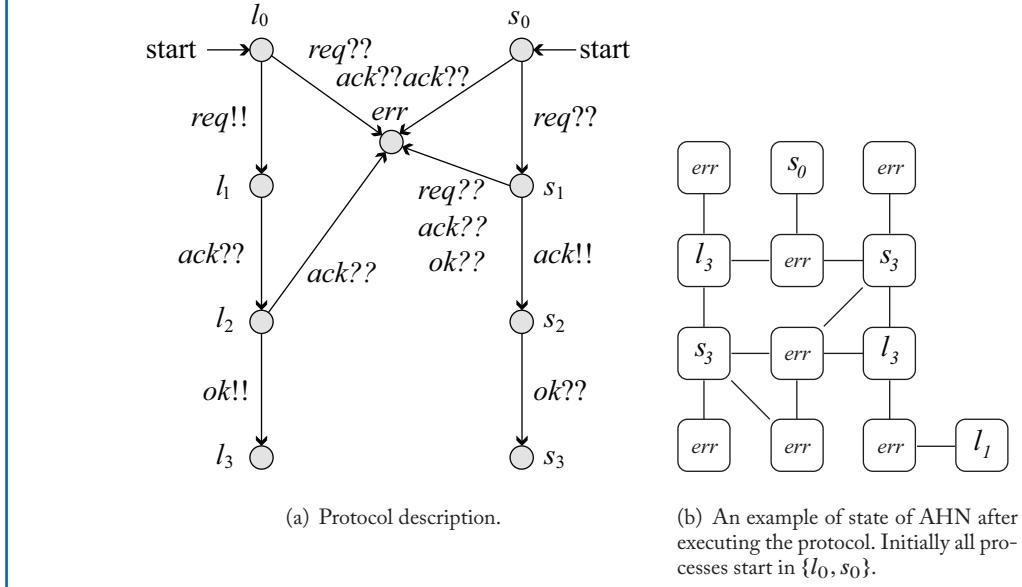


Figure 7.4: RAO protocol, the building block for protocols to find sub-graphs of a special form [Delzanno et al., 2010].

Proof idea. Note that process v can end in state s_3 only if it receives a broadcast message ok , hence there must be an adjacent process that is in state l_3 . By applying the RAO protocol backward, one can show that processes w, v can reach states l_3, s_3 only if they, starting from initial states l_0, s_0 , receive messages only from each other. Then, by applying the RAO protocol forwards to processes w, v starting in the initial states, one can show that if w, v reach l_3, s_3 then all their adjacent processes will end in state err . \square

Figure 7.4(b) shows an example of a system state after completion of the RAO protocol. Intuitively, the protocol links two adjacent processes (that is, a process in state s_3 is linked to a neighbor in state l_3) starting in states l_0, s_0 and sends all other adjacent processes into state err .

Undecidability of CoverAll. We are now ready to discuss the first undecidability result in the case of AHNs.

Theorem 7.6 [Delzanno et al., 2010] COVERAll for AHNs is undecidable.

Proof idea. To simulate a 2CM consider process template P that implements a two-phase protocol.

1. Find a sub-graph of the form in Figure 7.5(a), we call it “the controller with two lists.”
2. Simulate a 2CM.

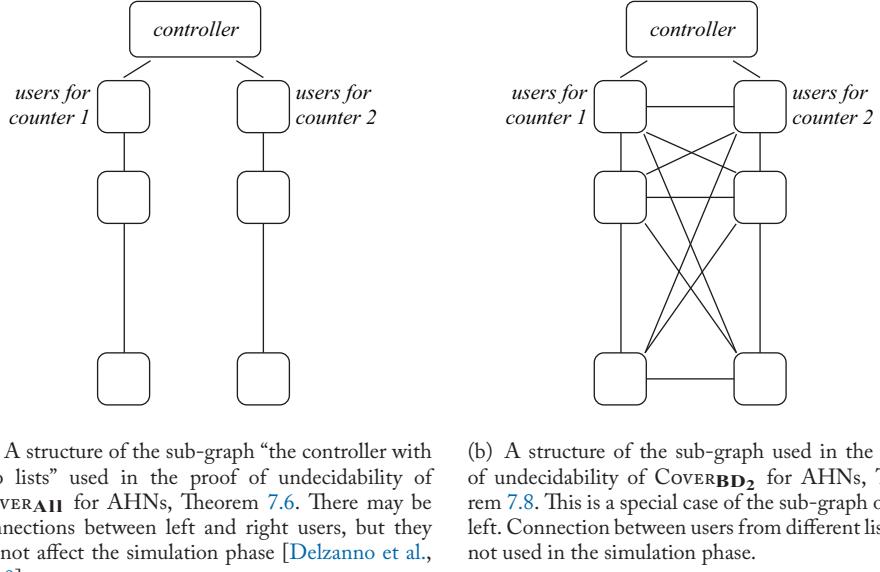


Figure 7.5: Sub-graphs used to simulate a 2CM.

The second phase starts only if the first phase succeeds. Using the protocol we will show that the 2CM halts if and only if there is a graph $G \in \text{All}$ such that system P^G reaches a state with some process in state *halt*.

1st phase: Finding a sub-graph “the controller with two lists.”

The first phase can succeed only if the connectivity graph has a sub-graph of a special form: a controller process connected to two lists of user processes. The protocol in Figure 7.6 tries to find such a sub-graph. Initial states are $\{R, C'_1, C''_1, C'_2, C''_2 | i = 1, 2\}$. Note that it is possible that after the protocol execution the connectivity graph is split into more than a single sub-graph each of the desired form, but these sub-graphs are isolated (nodes that connect them are in the error state and will not participate in the second phase). The transitions leading to the error state are not shown in the figure.

Roughly, the phase guesses a sub-graph of the desired form and then verifies it exists.

The phase can succeed only in executions in which the system starts in an initial state with at least one process in state R (call it the controller) that is connected to two lists of the form $C'_i - C''_i - C_i - C'_i - C''_i - C_i - \dots$ (call them list 1 and list 2 for $i = 1, 2$, and call the first nodes of two lists head 1 and 2). Note that non-determinism of the process initial state implies: even if there is a sub-graph “the controller with two lists,” it is still possible that the first phase fails because the initial system state is not of the desired form. But it is certain that if there is a desired sub-graph, then there is a desired initial system state.

112 7. AD HOC NETWORKS

The phase uses a modified RAO protocol that has the following differences. The original RAO protocol ensures that two adjacent processes that are initially in states l_0 and s_0 can reach states l_3 and s_3 while sending all other adjacent processes into state err . In contrast, the modified RAO protocol in Figure 7.6 describes several-steps protocol that links an R -process with a unique C'_1 -process (step 1), that C'_1 -process with a C''_1 -process (step 2), that C''_1 -process with a C_1 -process (step 3), and that C_1 -process with a C'_1 -process (similar to step 1 but the C_1 -process plays the role of an R -process).

Step 1 links an R -process with a unique adjacent C'_1 -process. All other processes adjacent to the R -process will go either into err state or they belong to another counter and do not change their states. All other processes adjacent to the C'_1 process will go either into err except those in C'' state.

Step 2 links a C'_1 -process with a unique adjacent C''_1 -process, while sending all others adjacent to C'_1 into err except the previously linked C_1 -process, and sending all others adjacent to C''_1 into err except those in state C_1 .

Step 3 links a C''_1 -process with a unique adjacent C_1 -process, while sending all others adjacent to C''_1 into err except the previously linked C'_1 -process, and sending all others adjacent to C_1 into err except those in state C'_1 .

In all steps a process that initiates linking with another process (in Step 1 – C_1 -process, in Step 2 – C'_1 -process, in Step 3 – C''_1 -process) non-deterministically chooses between linking with another process or sending $done_1$. Sending $done_1$ completes the initialization of list 1 – the message is transmitted back to the controller (an R -process), and the controller starts initializing list 2 in a similar way. If the initialization of both lists completes (thus the controller is not hanged waiting for a message from head 1 or 2), then the system state is of the form $0_1 - \dots - 0_1 - L - 0_2 - \dots - 0_2$, and the controller starts simulating the 2CM.

2nd phase: Simulating a 2CM. If the first phase succeeds, then there is at least one sub-graph of the form in Figure 7.5(a). Consider one such sub-graph. In the simulation phase the controller process simulates transitions of the control state of the 2CM, while user nodes encode values of counters (each user “stores” a bit, and the sum of bits in list i is equal to the value of counter i). To increase the value of counter 1 the controller broadcasts message inc_1 which is received by two head processes. Only head 1 reacts to the message: if it has its bit set, then it passes the message further along list 1, otherwise head 1 sets its bit and sends ack to acknowledge the message. When the controller receives ack , it continues simulation. “Decrease” and “test for zero” work similarly. This completes the description of the simulation phase.

The protocol ensures that the 2CM halts if and only if there is a system where the controller process can reach a halting state, which is an instance of **COVER_{II}** problem. \square

Example 7.7 Now we can relate the results in Figure 7.3, in particular Corollary 7.4 and Theorem 7.6, with the running example from Section 7.1. Since **COVER_{II}** for AHNs is undecidable, there is no single algorithm that can be used to solve all PMCP instances of the form (i)

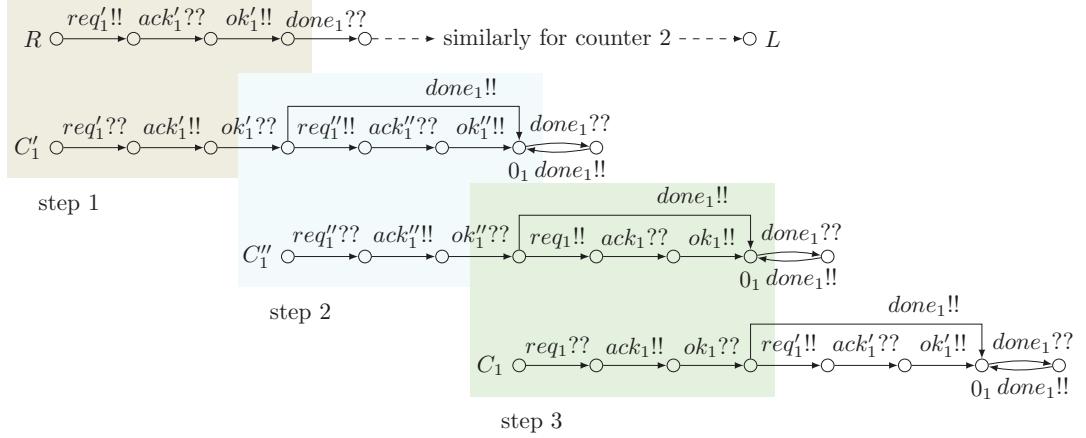


Figure 7.6: A part of the protocol to find a sub-graph “controller with two lists” used in Theorem 7.6. Only the part which looks for a list of processes that store the value of counter 1 is shown. The protocol can be naturally extended for both counters: after receiving $done_1$ the controller runs a similar protocol but with messages $req_2, ack_2 \dots$ instead of $req_1, ack_1 \dots$. Transitions to state err are omitted for readability [Delzanno et al., 2010].

from Section 7.1 (recall from Example 7.2 that the PMC in (i) is equivalent to an instance of $\text{COVER}_{\text{All}}$). In contrast, the corresponding PMCP for AHNs arranged in cliques is decidable by Corollary 7.4—thus, an algorithm that solves $\text{COVER}_{\mathcal{C}}$ can also be used to answer the PMC question in (ii).

Undecidability for \mathbf{BD}_k . Consider the case of a more restricted parameterized connectivity graph \mathbf{BD}_k . Since \mathbf{BD}_1 coincides with the set of all cliques, it follows from Corollary 7.4 that $\text{COVER}_{\mathbf{BD}_1}$ is decidable, and that $\text{REPEAT}_{\mathbf{BD}_1}$ and $\text{TARGET}_{\mathbf{BD}_1}$ are undecidable. For $k \geq 2$ $\text{COVER}_{\mathbf{BD}_k}$ becomes also undecidable.

Theorem 7.8 [Delzanno et al., 2011, Theorem 1] $\text{COVER}_{\mathbf{BD}_k}$, for AHNs, is undecidable for $k \geq 2$. Notice that this theorem does not directly follow from the previous one. We prove it differently from Delzanno et al. [2011], where the authors developed a separate protocol for \mathbf{BD}_k .

Proof idea. When one tries to prove undecidability of $\text{COVER}_{\mathbf{BD}_2}$ (other k s follow), the first natural question is “Can we reuse the protocol from All case?” It might happen that the sub-graph extracted from All (of the form in Figure 7.5(a)) is not present in \mathbf{BD}_k , which would mean that the protocol that was able to guess and then extract two lists of unbounded length from All will not be able to extract such lists from \mathbf{BD}_2 . I.e., when working on graphs in \mathbf{BD}_2 , the protocol from All case can guess the presence of a long list but it cannot complete its execution, because \mathbf{BD}_2 does not contain the list of such length.

114 7. AD HOC NETWORKS

This is not the case though, since the protocol for **All** does not extract two completely “isolated” lists (connected to the controller), but rather two lists that can have interconnections between nodes from different lists. In other words, **BD₂** has the sub-graph shown in Figure 7.5(b) that i) can be extracted by the protocol from **All** case, ii) is of the form needed by the second phase when simulating a given 2CM. Hence, the protocol from **All** can be reused for **BD₂** and **Cover_{BD_k}** for $k \geq 2$ is undecidable. \square

It is not difficult to show that undecidability of **Cover_{BD_k}** implies undecidability of **Repeat_{BD_k}** and **Target_{BD_k}**, hence:

Corollary 7.9 [Delzanno et al., 2011, Esparza et al., 1999] **Repeat_{BD_k}**, **Target_{BD_k}**, for AHNs, are undecidable for $k \geq 1$.

Undecidability of Target_{BP_k}, Repeat_{BP_k, Target_{BPC_k, Repeat_{BPC_k. Now consider even a more restricted parameterized connectivity graph **BP_k**. Restricting the graph makes **Cover** decidable (see Theorem 7.19), but **Repeat** and **Target** are still undecidable:}}}

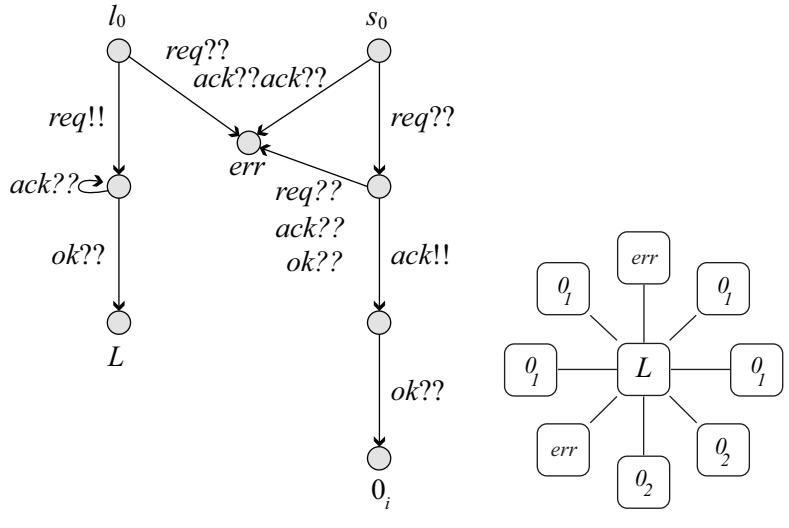
Theorem 7.10 [Delzanno et al., 2010, Theorem 6] **Target_{BP_k}** and **Repeat_{BP_k}**, for AHNs, are undecidable for $k \geq 2$.

Proof idea. Here we give an idea how to reduce the halting problem of 2CMs to **Target_{BP₂}**. The idea can be adapted to **BP_k** with $k > 2$. Consider a process template that implements a two-phase protocol:

1. Find a sub-graph of a form “star with a controller in the center.”
2. Simulate the 2CM.

The first phase of the protocol is simple in comparison to that of the previous theorems and is shown in Figure 7.7(a). If the first phase succeeds, then nodes that are not in the error state form a graph of the form “star with a controller in the center” with a node in the center called the controller.

In the second phase the controller simulates 2CM, while user nodes store counters values (each user “stores” a bit, the total number of bits being equal to the value of a counter). In the second phase ack_i, ok are different from those used in the first phase. To increase the value of counter i the controller broadcasts $inc_i!!$. If there is a user process in state 0_i , then it responds with $ack_i!!$ and goes to state 1_i . The controller then in its turn acknowledges it with $ok_i!!$ which prevents other processes from sending $ack_i!!$. If the controller though receives several $ack_i!!$ before it manages to send the acknowledgement then it moves into state err and never leaves it. Operation “decrease” works in a similar way. Instead of simulating operation “test for zero” for counter i , the controller non-deterministically guesses the result, and if the guess is zero, then the controller broadcasts $zero_i!!$ and continues simulation. If the guess is wrong and there is a user in state 1_i , then that user moves into state err .



(a) A modified RAO protocol to find the sub-graph “star with a controller in the center” ($i \in 1, 2$).

(b) An example of state of an AHN with the graph from **BP₂** after executing the protocol on the left.

Figure 7.7: Finding a star-like sub-graph in graph from **BP₂** [Delzanno et al., 2010].

On reaching a halting state the controller sends all users except those in state *err* to a special state *halt* and goes there itself. Therefore, 2CM halts if and only if there is a system where all processes reach *halt*, which is an instance of TARGET_{BP₂}. \square

Recall that if each clique in a graph from \mathbf{BPC}_k is replaced with a single node, then the resulting graph belongs to \mathbf{BP}_k . Hence in order to show undecidability of $\text{TARGET}_{\mathbf{BPC}_k}$ and $\text{REPEAT}_{\mathbf{BPC}_k}$ we modify the protocol used in the previous proof and add an initial phase in which in each clique a single representative node is chosen, and the rest of the protocol is unchanged. Hence (\mathbf{BPC}_0 coincides with the parameterized clique and was considered in Corollary 7.4):

Corollary 7.11 $\text{TARGET}_{\mathbf{BPC}_k}$ and $\text{REPEAT}_{\mathbf{BPC}_k}$, for AHNs, are undecidable for $k \geq 0$.

Undecidability of Cover_{daAll}. Finally, we state the result of Abdulla et al. [2013a] for the case of directed acyclic graphs:

Theorem 7.12 [Abdulla et al., 2013a, Theorem 1] COVER_{daAll} for AHNs is undecidable.

Proof idea. The previous proofs used the fact that the parameterized graphs are undirected: the basic RAO protocol in Figure 7.13(c) assumes the ability of a process to send and receive broadcast messages to and from all its adjacent nodes. If this is not the case as can be in directed graphs,

116 7. AD HOC NETWORKS

the RAO protocol's main property ("isolation of pairs of processes," Lemma 7.5) does not hold. Furthermore, the simulation construction described in the proof of Theorem 7.6 breaks in the general case of directed graphs, because the controller is not able to get the feedback about its commands from the user processes. These facts make adaption of the standard proof technique (the reduction from the halting problem for 2CMs to COVER) not easy.

Hence, for the case of parameterized graphs **daAll**, Abdulla et al. [2013a] considered a different undecidable problem called TRANSD. Informally, TRANSD takes as input two automata A and B , and one transducer T , all with the same alphabet (transducer is an automaton that outputs a symbol in each transition). TRANSD outputs "Yes" iff there is $i \in \mathbb{N}_0$ and a word w accepted by A such that i successive applications of the transducer T to the word w results in a word accepted by B .

The authors briefly show in the paper that TRANSD is undecidable. Indeed, a transducer can compute one step of a Turing machine, and thus iterating this transducer simulates arbitrarily many steps of that Turing machine.

To reduce TRANSD to COVER_{**daAll**}, the authors build a process template that simulates a given automata and a transducer. Initially, every process decide whom it simulates – A , B or T . For the simulation to be correct, it is necessary that the processes will form a graph of the form $A \rightarrow T \rightarrow \dots \rightarrow T \rightarrow B$, i.e., the first process simulates automaton A followed by a (possibly zero) number of processes simulating transducer T , and then a process that simulates automaton B . Initially every A -process sends a special initiating message intended for a T - or B -process—on receiving it every T -process sends the same message. Every A -, T -, B - process on receiving initiating message twice goes into error state, it also goes into the error if received a message not intended for them or received two messages with no sending in between. After sending the initiating message the A -process starts transmitting a word accepted by A , letter by letter, followed by the end marker. The T -process receives a letter of the word and sends the transduced one further, and so on, to the B -process. The B -process receives a letter and simulates transitions of the automaton B . The B -process goes into state *halt* if it receives the end marker and it is in an accepting state of the automaton.

The control state of COVER consists of a single state *halt*. The construction above ensures: there is an AHN P^G with $G \in \mathbf{daAll}$ where a B -process can go into the control state, if and only if some number of applications of T to a word of automaton A results in a word of automaton B . \square

7.5.2 DECIDABILITY RESULTS FOR (NON-LOSSY) AHNS

In the previous section, in the proofs of undecidability of COVER_{**All**} (Theorem 7.6) and COVER_{**BD_k**} (Theorem 7.8) we used "lists" of user nodes to store the values of counters of a 2CM. In the general case there is no bound on counter values, so the proofs only work if we can find lists of unbounded length in the parameterized connectivity graph. Thus, the undecidability proofs break for parameterized connectivity graphs in which all graphs have only "lists" of a bounded length,

or, in other words, if we restrict the length of simple paths on which a message in the system can travel. In the literature several such parameterized connectivity graphs are considered, namely, \mathbf{BP}_k , \mathbf{BPC}_k , and \mathbf{dabP}_k .

This section proves Theorem 7.19 that states decidability of COVER on \mathbf{BP}_k , \mathbf{BPC}_k and on \mathbf{dabP}_k , thus covering all decidability results from Figure 7.3 for (non-lossy) AHNs.

The proofs of decidability of COVER of AHNs on these parameterized connectivity graphs use the machinery of well structured transition systems (WSTS) (see Section 3.2.1). The proofs have a similar structure for all the graphs except \mathbf{dabP}_k : Abdulla et al. [2013a] first reduced COVER on \mathbf{dabP}_k to COVER on inverted trees—that is, trees that contain a node v such that all other nodes have a directed path to v —of bounded depth.

This section is organized as follows.

1. We define the induced sub-graph relation \leq_{is} on directed graphs.
 2. Lemma 7.13 proves that COVER on directed acyclic bounded path graphs (\mathbf{dabP}_k) can be reduced to COVER on bounded inverted trees (\mathbf{BIT}_k , defined later).
 3. Lemma 7.14 and 7.15 prove that \leq_{is} is a well quasi-order on \mathbf{BPC}_k , \mathbf{BP}_k , \mathbf{BIT}_k .
 4. We define transition system M , which for given P , G , represents the behavior of all instances of the parameterized AHN P^G .
 5. Lemma 7.18 proves that the transition system M for AHNs on \mathbf{BP}_k , \mathbf{BPC}_k , \mathbf{BIT}_k is a WSTS with respect to \leq_{is} (the proof uses Lemmas 7.14 and 7.15, and also intermediate Lemmas 7.16 and 7.17).
 6. Finally, using Lemmas 7.13 and 7.18, Theorem 7.19 proves decidability of COVER for AHNs on \mathbf{BP}_k , \mathbf{BPC}_k , and \mathbf{dabP}_k .
 7. We describe the parameterized model checking algorithm that solves COVER for AHNs on \mathbf{BP}_k , \mathbf{BPC}_k , and \mathbf{dabP}_k , and illustrate how it works.
- 1. The induced sub-graph relation \leq_{is} .** For graphs G_1, G_2 and labeling functions $s_1 : V_1 \rightarrow Q$ and $s_2 : V_2 \rightarrow Q$ with a set of labels Q define: $(G_1, s_1) \leq_{is} (G_2, s_2)$ iff there is a label preserving injection $h : V_1 \rightarrow V_2$ such that $(v, w) \in E_1 \iff (h(v), h(w)) \in E_2$. This is different from the sub-graph relation which preserves labels but allows new edges, i.e., $(h(v), h(w)) \in E_2$ does not necessarily imply $(v, w) \in E_1$. Figure 7.8 gives an example.
- 2. Reducing Cover $_{\mathbf{dabP}_k}$ to Cover $_{\mathbf{BIT}_k}$.** Notice that the induced sub-graph relation \leq_{is} is not a well quasi-order on \mathbf{dabP}_k (an example of infinite decreasing chain is in Abdulla et al. [2013a, Section 8]). This is why we first need to reduce Cover $_{\mathbf{dabP}_k}$ to COVER on inverted trees of bounded depth for which \leq_{is} is a well quasi-order (Lemma 7.15). Let \mathbf{BIT}_k be a parameterized connectivity graph consisting of all inverted trees of depth k . Informally, an inverted tree is a



Figure 7.8: Example for the induced sub-graph relation \leq_{is} .

tree with all edges pointing toward the root. For the formal definition see, for example, [Abdulla et al., 2013a].

Then the following reduction from $\text{COVER}_{\text{daBP}_k}$ to $\text{COVER}_{\text{BIT}_k}$ holds.

Lemma 7.13 [Abdulla et al., 2013a, Theorems 2 and 5] Given a decision procedure for $\text{COVER}_{\text{BIT}_k}$, one can construct a decision procedure for $\text{COVER}_{\text{daBP}_k}$.

Proof idea. For a given graph in daBP_k we build an inverted forest of depth k that preserves all the paths of the original graph but has new nodes (example in Figure 7.9).

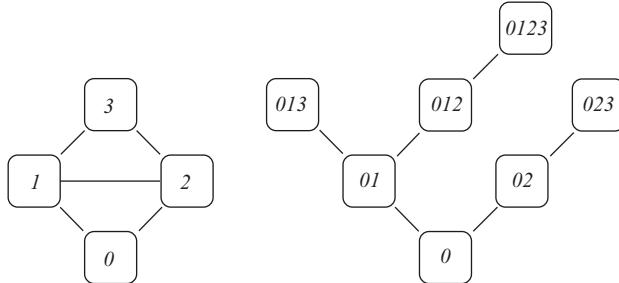


Figure 7.9: [Abdulla et al., 2013a] Directed acyclic graph and a corresponding inverted forest used in Lemma 7.13.

Then: if a process state is reachable in the AHN on a graph from daBP_k , then this state is also reachable in the AHN on the inverted forest of depth k . The idea is to simulate the behavior of the original nodes (e.g., in Figure 7.9 all nodes with names starting with 0 will repeat the behavior of node 0). The other direction—if a process state is reachable in an AHN on an inverted forest of depth k , then it is reachable in the AHN on a graph from daBP_k —is trivial since any inverted forest of depth k is in daBP_k .

Recall that a forest consists of trees and processes do not communicate with each other. Hence, a process can reach a control state in the AHN on an inverted forest if and only if a process from one of the trees can reach this state. This means that instead of considering COVER on all inverted forests of depth k we can consider COVER on all trees of those forests,

i.e., COVER on all inverted trees of depth k . Finally, this means that COVER_{**daBP_k**} is equivalent to COVER_{**BIT_k**}. \square

3. \leq_{is} is a well-quasi-order on **BP_k**, **BPC_k**, **BIT_k**.

Lemma 7.14 [Delzanno et al., 2011, Lemma 2] For fixed $k \in \mathbb{N}$, the induced sub-graph relation \leq_{is} is a well quasi-order on labeled graphs from **BPC_k**.

Proof idea. The proof extends that of Ding [1992, Theorem 2.1] for the case of **BP_k**. The crucial property of **BPC_k** (and of **BP_k**) is that in any graph from **BPC_k** there is a vertex such that its removal splits the graph into non-connected graphs from **BPC_{k-1}**. This property, together with Higman's lemma [Higman, 1952] (informally: if \leq is a well quasi-order on Q , then the component-wise variation \leq^* is a well quasi-order on Q^*), leads to an inductive proof. \square

BP_k \subset **BPC_k** for any $k \geq 1$, hence by the lemma \leq_{is} is a well quasi-order on **BP_k** too (the case of $k = 0$ is trivial).

Abdulla et al. [2013a] introduced a special class of multi-sets that can encode inverted trees of depth k , and then prove that these sets are well quasi ordered and so are inverted trees of depth k . Here, instead, we show that inverted trees of depth k are well quasi-ordered with the induced sub-graph relation \leq_{is} .

Lemma 7.15 For a fixed $k \in \mathbb{N}$, the induced sub-graph relation \leq_{is} is a well quasi-order on labeled graphs from **BIT_k**.

Proof idea. We use the fact that \leq_{is} is a well quasi-order on labeled **BP_k** [Ding, 1992, Theorem 2.1].

First, any inverted tree of depth k can be uniquely up to isomorphism mapped to a $3k$ -bounded path graph. For example, the inverted tree $a \rightarrow b$ maps to $a - v - w - b$, where v, w are two auxiliary nodes to help encode edge directions².

Take any infinite sequence G_1, G_2, \dots of inverted trees of depth k . We will show that there is $i < j$ with $G_i \leq_{is} G_j$.

Consider the sequence G'_1, G'_2, \dots of encoding graphs. They are from $3k$ -bounded path graphs which are well quasi ordered with \leq_{is} [Ding, 1992, Theorem 2.1], and therefore the sequence has $i < j$ with $G'_i \leq_{is} G'_j$.

Thus, for original graphs, $G_i \leq_{is} G_j$ also holds: by induction we can show that the induced sub-graph relation preserves exact paths, hence for any path $a - v - w - b$ in G'_i there is a path $h(a) - h(v) - h(w) - h(b)$ in G'_j , hence for edge $a \rightarrow b$ in G_i there is edge $h(a) \rightarrow h(b)$ in G_j . The other direction (for any edge $h(a) \rightarrow h(b)$ there is edge $a \rightarrow b$) can be proven similarly.³

\square

²This encoding was suggested to us by authors of [Abdulla et al., 2013a], in particular by Othmane Rezine.

³Here, h is a label preserving injection $V(G'_i) \rightarrow V(G'_j)$ as required by the definition of \leq_{is} , we also assume that for any i : $V(G'_i) = V(G_i) \cup \text{auxiliary nodes}$, and we reused h to show that $G_i \leq_{is} G_j$.

120 7. AD HOC NETWORKS

4 & 5. Parameterized AHNs on \mathbf{BP}_k , \mathbf{BPC}_k , \mathbf{BIT}_k are WSTSs wrt. \leq_{is} . For given parameterized AHN $\overline{P}^{\mathbf{G}}$ define the transition system M (in Lemma 7.18 we prove it is a WSTS).

- The set of states consists of all pairs (G, s) with $G \in \mathbf{G}$, $s \in S$ where S is the set of states of P^G , and the set I of initial states $I = \{(G, s_0) \mid G \in \mathbf{G}, s_0 \in S_0\}$. Notice that I is downward closed with respect to \leq_{is} .
- The transition relation contains $(G, s) \rightarrow (G, s')$ iff there is a transition $s \rightarrow s'$ in P^G .

Intuitively, the transition system M is an infinite state system that represents all instances of a given parameterized AHN $P^{\mathbf{G}}$.

Now we can state the following.

Lemma 7.16 [Delzanno et al., 2010, Lemma 2]⁴ Fix $\overline{P}^{\mathbf{G}}$, and let M be defined as above. Then the induced sub-graph relation \leq_{is} is monotonic with respect to the transition system M .

Proofidea. Let $(G, s) \leq (G', s')$ for some states s, s' in S and G, G' in \mathbf{G} (where S is the set of states of $\overline{P}^{\mathbf{G}}$). Consider the case when $(G, s) \rightarrow (G, t)$ happens via a broadcast by process v in system \overline{P}^G . Then there is a process v' of $\overline{P}^{G'}$ which is in the same state as process v , and v' is connected to processes in the same states as neighbors of v . Then if the process v' in $\overline{P}^{G'}$ makes the same broadcast action as v in \overline{P}^G then all the neighbors of v' in G' behave the same way as neighbors of v in G , and possibly neighbors of v' in $G' \setminus G$ also change state. \square

Recall from Section 3.2: for a given set C of states of transition system M , $Pred(C)$ is defined as the union of C and all one-step predecessors of C ; a basis of an upward closed set D is set B s.t. $D = \uparrow B$. Now we are ready to state:

Lemma 7.17 [Abdulla et al., 2013a, Delzanno et al., 2010, 2011]

Fix $k \in \mathbb{N}_0$, $\overline{P}^{\mathbf{G}}$ where \mathbf{G} is one of \mathbf{BP}_k , \mathbf{BPC}_k , \mathbf{BIT}_k , and let M be the transition system defined as before. Then given a basis of an upward closed set C , one can compute a basis of $Pred(C)$ for the transition system M .

Proofidea. A variation of the algorithm is in Algorithm 1. The authors do not provide the proof of the algorithm correctness, but the insight is: to compute $Pred(\uparrow B)$ for some finite $B = ((G_1, s), \dots, (G_i, s_i), \dots, (G_k, s_k))$, it is enough to compute all predecessors of all (G_i, s_i) in B , and compute all predecessors of all (G_i^+, s_i^+) resulting from (G_i, s_i) by adding a single node making a broadcast transition. The termination follows from the finiteness of iterated objects. \square

⁴ The original lemma is for graphs from \mathbf{BP}_k , but the proof applies to general graphs as well.

Algorithm 1: BPred: Computing a finite basis of a predecessor (Lemma 7.17).

Input: $P, G \in \{\mathbf{BP_k}, \mathbf{BPC_k}, \mathbf{BIT_k}\}$, a finite set T of states of the transition system M

Output: a basis of $\text{Pred}(\uparrow T)$

def BPred(P, G, T):

```

     $B = T$ 
    for  $(G, s) \in T$  :
        for  $q' \xrightarrow{\text{act}} q$  of  $P$  :
             $A = \text{pred1}(G, s, q' \xrightarrow{\text{act}} q)$ 
            add to  $B$  all  $(G, s) \in A$  that are not in  $\uparrow B$ 
            if  $q' \xrightarrow{\text{act}} q$  is a broadcast send transition :
                 $T^+ = \{(G^+, s^+) \mid \text{it is } (G, s) \text{ with one new node in state } q \text{ and } G^+ \in \mathbf{G}\}$ 5 for
                 $(G^+, s^+) \in T^+$  :
                     $A = \text{pred1}(G^+, s^+, q' \xrightarrow{\text{act}} q)$ 
                    add to  $B$  all  $(G, s) \in A$  that are not in  $\uparrow B$ 
    return  $B$ 

```

def pred1($G, s, q' \xrightarrow{\text{act}} q$):

```

     $A = \emptyset$ 
    for  $v \in G$  with  $s(v) = q$  :
        if  $q' \xrightarrow{\text{act}} q$  is internal :
            add  $(G, s')$  to  $A$  where  $s'$  is the predecessor of  $s$  corresponding to  $q' \xrightarrow{\text{act}} q$ 
        elif  $q' \xrightarrow{\text{act}} q$  is broadcast send  $q' \xrightarrow{a!!} q$  :
            if process  $v$  has a neighbor6 in state  $r$  and  $r' \xrightarrow{a??} r$  for some  $r$  :
                continue
             $\{v_1, \dots, v_k\} =$  the neighbors of  $v$  that received  $a$  (for every  $v_i \exists r'_i \xrightarrow{a??} s(v_i)$ )7
            for subset  $\in 2^{\{v_1, \dots, v_k\}}$  :
                 $S' = \{s' \mid s' \text{ is a predecessor of } s \text{ where } v \text{ synchronizes only with subset}$ 8
                add  $(G, s')$  to  $A$  for every  $s' \in S'$ 
    return  $A$ 

```

⁵To optimize, consider only G^+ s where the new node is connected to at least one node that will be able to receive the broadcast.
⁶“ v has a neighbour v' ” means there is an edge from v to v' .

⁷For each v_i there might be more than one r'_i with $r'_i \xrightarrow{a??} s(v_i)$.

⁸I.e., $\forall v_i \in \text{subset} : s'(v_i) \xrightarrow{a??} s(v_i)$ is a transition of P , $\forall v' \in G \setminus \{v, v_i\} : s'(v') = s(v'), s'(v) = q'$.

122 7. AD HOC NETWORKS

Lemma 7.18 Fix $k \in \mathbb{N}_0$, \overline{P}^G where G is one of \mathbf{BP}_k , \mathbf{BPC}_k , \mathbf{BIT}_k , and let M be the transition system defined as before. Then M is a well structured transition system with respect to \leq_{is} .

Proof. M and \leq_{is} satisfy all items of the definition of a WSTS wrt. \leq_{is} (Section 3.2.1):

- \leq_{is} is a well quasi-order on states of M (Lemmas 7.14 and 7.15),
- \leq_{is} is monotonic with respect to M (Lemma 7.16), and
- a basis of $Pred(\uparrow B)$ for a given finite set B of states of M is computable (Lemma 7.17)

□

6. **Cover is decidable.** Finally, we are ready to prove the main result of this section.

Theorem 7.19 [Abdulla et al., 2013a, Delzanno et al., 2010, 2011] COVER is decidable for AHNs on i) \mathbf{BPC}_k , ii) \mathbf{BP}_k , iii) \mathbf{daBP}_k .

Proof idea. Consider the case of \mathbf{BPC}_k , the cases of \mathbf{BP}_k and \mathbf{BIT}_k (which implies the result for \mathbf{daBP}_k by Lemma 7.13) is similar. For simplicity assume that the set of control states given to COVER consists of a single state.

We reduce $\text{COVER}_{\mathbf{BPC}_k}$ to the coverability problem for WSTSs and then apply Theorem 3.3.

Fix process template $P, k \in \mathbb{N}_0$, and a single control state c in $\text{COVER}_{\mathbf{BPC}_k}$. For given P, k build the transition system M defined as before. Consider the instance of coverability problem for WSTS M with control states set $\{(G_1, s) \mid s(1) = c\}$ where G_1 is the graph with a single vertex. M is a WSTS (by Lemma 7.18) with downward closed initial states, hence by Theorem 3.3 we can compute the answer to the coverability problem for M which is equal to the answer to $\text{COVER}_{\mathbf{BPC}_k}$ for $P^{\mathbf{BPC}_k}$. Hence, $\text{COVER}_{\mathbf{BPC}_k}$ for AHNs is decidable. □

7. **Putting it all together: the parameterized model checking algorithm.** Let us develop the parameterized model checking algorithm that solves COVER_G for AHNs for $G \in \{\mathbf{BPC}_k, \mathbf{BP}_k, \mathbf{BIT}_k\}$. The soundness and completeness of the algorithm will follow from Theorem 7.19 and Theorem 3.3. The algorithm computes predecessors of C using procedure BPred from Algorithm 1 until the fixpoint is reached, and checks if predecessors has a non-empty intersection with the initial states. Algorithms of such form are called set saturation algorithms in Finkel and Schnoebelen [2001].

For simplicity assume that the set of control states given to COVER consists of a single state called c , thus $C = \{c\}$. Also, let G_c denote the graph with a single node in state c .

Algorithm 2: Solving COVER.

```

Input:  $P, c, G \in \{\mathbf{BP}_k, \mathbf{BPC}_k, \mathbf{BIT}_k\}$ 
Output: the answer to COVERG for  $P, c, G$ 
 $b\_reach = \emptyset$ 
 $b\_reach' = \{G_c\}$ 
while  $b\_reach \neq b\_reach'$  :
     $b\_reach = b\_reach'$ 
     $b\_reach' = \text{BPred}(P, G, b\_reach)$ 
    if  $b\_reach'$  contains an initial system state :
        | return YES
    return NO
    
```

Example 7.20 Let us apply Algorithm 2 to the process template P_{re} from the running example in Figure 7.1, the parameterized graph \mathbf{BP}_2 (“stars”), and for different control states C .

First, consider $C = \{err\}$. Thus, we answer PMC question (i) from Section 7.1 specialized to \mathbf{BP}_2 : “In AHNs composed of processes of P_{re} , is there a graph from \mathbf{BP}_2 and a process that reaches err ? ”

- The first iteration gives: $\text{BPred}(P, \mathbf{BP}_2, \{G_{err}\}) = \{err, init - l'\}$ ⁹.
- The second iteration gives: $\text{BPred}(P, \mathbf{BP}_2, \{err, init - l'\}) = \{err, init - l'\}$. Fixpoint!
- $\{err, init - l'\}$ does not contain an initial system state, hence return NO.

Now consider $C = \{hub\}$ and thus the question is “In AHNs composed of processes of P_{re} , is there a graph from \mathbf{BP}_2 and a process that reaches hub ? ”

- The first iteration gives: $\text{BPred}(P, \mathbf{BP}_2, \{G_{hub}\}) = \{hub, init - u, init - u', u - u'\}$.
- For simplicity, consider only labeled graph $init - u'$ calculated in the previous item: $\text{BPred}(P, \mathbf{BP}_2, \{init - u'\}) = \{init - u', init - init - init\}$. The set contains initial system state $init - init - init$, hence return YES.

7.6 DECIDABILITY RESULTS FOR LOSSY AD HOC NETWORKS

Recall from Section 7.2 that in LAHNs a broadcast message can be non-deterministically lost by some of the receivers.

⁹Here, a denotes the graph consisting of a single node labeled a . Also, $a - b$ denotes the graph consisting of two connected vertices labeled a and b . Finally, $a - b - c$ denotes the graph which is a pipeline of three nodes labeled a, b , and c , respectively.

124 7. AD HOC NETWORKS

The following two theorems were proved for *mobile* ad hoc networks, but as shown in Delzanno et al. [2012b, Proposition 1], mobile ad hoc networks are equivalent to lossy ad hoc networks on cliques.

Theorem 7.21 [Delzanno et al., 2010, Corollary 2] $\text{COVER}_{\mathbf{C}}$, $\text{REPEAT}_{\mathbf{C}}$, $\text{TARGET}_{\mathbf{C}}$ are decidable for LAHNs.

Proof idea. The problems can be reduced to decidable problems for Petri nets (or equivalently to vector addition systems) described in Chapter 3. The reduction works only for systems arranged in cliques, so does not apply to \mathbf{BP}_k , \mathbf{BD}_k , \mathbf{BPC}_k , etc.

To reduce the problems for LAHN to decidable problems for Petri nets the authors suggest the construction of the Petri net shown in Figures 7.10 and 7.11 (the construction depends on the process template of a LAHN under consideration). The construction ensures that the answer to a given LAHN problem is YES if and only if the answer to a corresponding problem for the Petri net is YES.

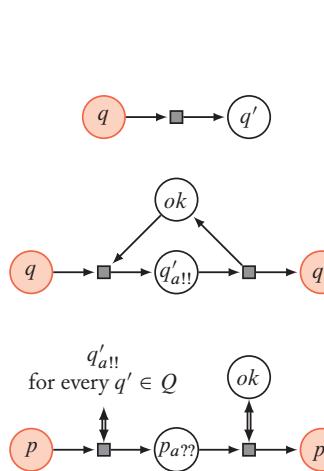


Figure 7.10: Top to bottom, Petri net constructions for transitions: internal, broadcast-send $q \xrightarrow{a!!} q'$, and receive $p \xrightarrow{a??} p'$.

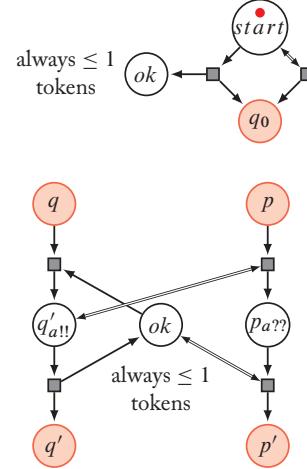


Figure 7.11: On the top—the initialization, on the bottom—illustration of broadcast synchronization $q \xrightarrow{a!!} q'$ and $p \xrightarrow{a??} p'$.

Assuming that COVER , REPEAT , TARGET has control sets with a single state, the reductions are the following.

- $\text{COVER}_{\mathbf{C}}$ is reduced to the coverability for Petri nets: the control configuration has “1” in a place that corresponds to a control state, and “0” in all other places.
- $\text{REPEAT}_{\mathbf{C}}$ is reduced to the repeated coverability for Petri nets: the control configuration is defined similarly to the previous case.

7.6. DECIDABILITY RESULTS FOR LOSSY AD HOC NETWORKS 125

- TARGET_C is reduced to the reachability for Petri nets, but the Petri net constructed according to Figures 7.10 and 7.11 additionally has:
 - place end ,
 - transition from ok to end , and
 - transition from end, q to end for every state q of the process template.

Then the control configuration is: “1” in place end , and “0” in all other places.

Now let us look how the reduction works.

Direction “Run in the LAHN \rightarrow computation in the Petri net”: take a run in the LAHN that satisfies a given problem property, let us build a run in the Petri net that also satisfies the corresponding property. First, the Petri net builds an initial state of the LAHN (Figure 7.11, top): the Petri net “pumps” an arbitrary number of tokens from $start$ into place q_0 , then the token from $start$ moves into ok . At this moment the configuration of the Petri net (with n tokens in q_0) corresponds to the initial state of the LAHN with n processes.

After the initial configuration is built, the Petri net starts simulating transitions of the LAHN. Consider the case of a broadcast transition: send $q \xrightarrow{a!!} q'$ and receive $p \xrightarrow{a??} p'$ (Figure 7.11, bottom). Suppose there is a process in state q in the LAHN and some number of processes in state p – in the Petri net this corresponds to a token in place q , and a number of tokens in p (the number of tokens in p is equal to the number of processes in state p). Let the q -process and m processes in state p take the broadcast synchronized transition—in the Petri net this corresponds to a sequence of transitions that moves a single token from q to q' and m tokens from p to p' (and no tokens are left in $p_{a??}$). This completes the description of the simulation of the broadcast transition. The simulation ensures that the computation of the Petri net satisfies the corresponding property (over Petri net configurations).

Direction “Faithful computation in the Petri net \rightarrow run in the LAHN.” We call *faithful computation* a computation in the Petri net that does not leave tokens in place $p_{a??}$ between broadcast simulations. Then the proof of this direction follows from two claims.

- (i) If there is a computation that (repeatedly) covers the corresponding control configuration mentioned in the reductions before, then there is a faithful computation in the Petri net that does this,
- (ii) Any faithful computation of the Petri net has a corresponding run in the LAHN.

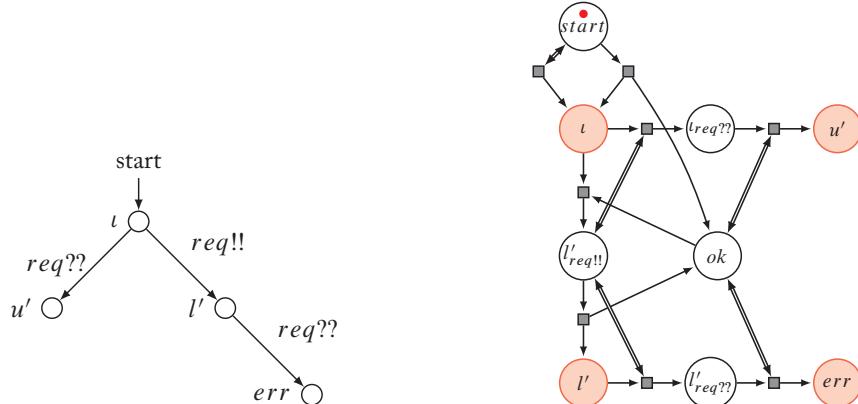
The intuition behind item (i) is as follows. If in a non-faithful computation there are tokens left in state $p_{a??}$, then in a faithful corresponding computation we move all the tokens from $p_{a??}$ to state p' and do not move them until the tokens in $p_{a??}$ get moved in the original non-faithful computation. The second item (ii) follows from the observation that any faithful transition of the Petri corresponds to some transition in the LAHN.

126 7. AD HOC NETWORKS

Finally, the decidability of $\text{COVER}_{\mathbf{C}}$, $\text{TARGET}_{\mathbf{C}}$, $\text{REPEAT}_{\mathbf{C}}$ for LAHNs follows from the decidability of corresponding problems for Petri nets. \square

Example 7.22 Consider process template in Figure 7.12(a) which is a part of the running example process template P_{re} in Figure 7.1 from Section 7.1. Constructing the Petri net according to Figures 7.10 and 7.11 gives the net in Figure 7.12(b).

Consider PMC question (i) from Section 7.1 adapted to LANHs: “Is there a LAHN build from process template in Figure 7.12(a) that reaches state $err?$.” It is easy to see that two tokens in the initial state ι are enough to cover state err in the Petri net: consider computation where, first, one token moves from ι to ι' , and then both tokens move: the token in ι' moves to err while the token from ι moves to ι' . Hence, the answer to our PMC question is YES.



(a) Part of the running example template.

(b) Petri net constructed from the protocol on the left.

Figure 7.12: Example of the conversion of a protocol into a Petri net simulating LAHNs.

Note on complexities. The proof of Theorem 7.21 reduces $\text{COVER}_{\mathbf{C}}$ and $\text{TARGET}_{\mathbf{C}}$ for LAHNs to EXPSPACE-complete and -hard problems on Petri nets, and does not provide the lower bound. In the follow-up paper Delzanno et al. [2012a] explore the complexities. We state their results without providing the proofs.

Theorem 7.23 [Delzanno et al., 2012a, Theorem 1 and 2] For LAHNs $\text{COVER}_{\mathbf{C}}$ is P-complete and $\text{TARGET}_{\mathbf{C}}$ is NP-complete (with respect to the size of the process template).

7.7 DISCUSSION

In ad hoc networks, the nature of the parameterized connectivity graph separates the decidable cases from undecidable. [Delzanno et al. \[2010, 2011\]](#) gave the first characterization of those graphs on which COVER is decidable (\mathbf{BPC}_k) and on which it is undecidable (\mathbf{BD}_k). Figure 7.13 illustrates the decidability boundary for this problem. If we consider the more expressive problems

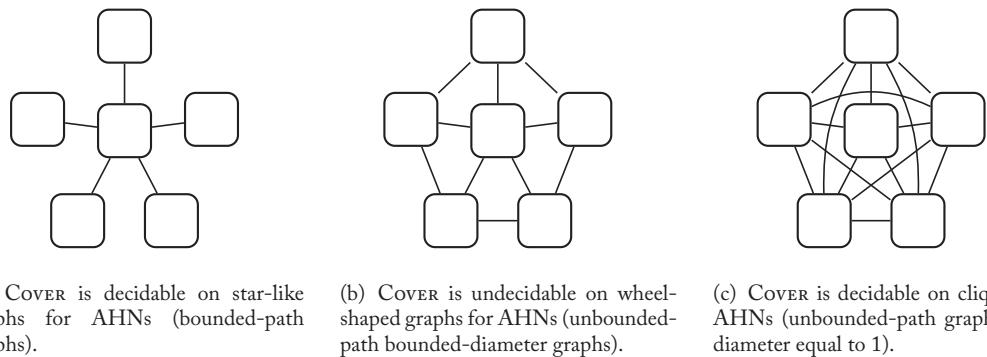


Figure 7.13: The decidability boundary for Cover for AHNs.

REPEAT or TARGET in ad hoc networks without losses, then we get undecidability for all the graphs considered in this chapter.¹⁰ Finally, in ad hoc networks with lossy broadcast transitions, all these problems are decidable. Table 7.1 summarizes the decidability results for ad hoc networks.

Decidability proof techniques in both non-lossy and lossy cases are based on well-structuredness of the systems, and do not exhibit cutoffs that are independent of the process template as in Chapter 4.

Undecidability proofs, similarly to other chapters, reduce from the halting problem for 2CMs. Thus, for a given 2CM, the proofs build a protocol that simulates its execution. In contrast to other chapters, the exact connectivity graph is unknown *a priori* (what is known is a class to which the graph belongs to), so the protocols for simulating a given 2CM have two steps. In the first step they ensure the right form of the connectivity graph, and the second step does the simulation relying on the particular form of the graph. Another difference is the case of directed acyclic graphs (Theorem 7.12) where the information from process to process flows in one direction only. Thus the controller that simulates the control of a given 2CM cannot receive the feedback from processes it broadcasts messages to. In this case the undecidability proof uses the fact that TMs can be simulated by iterating finite-state transducers.

¹⁰Possibly except for directed parameterized connectivity graph \mathbf{dabP}_k .

Table 7.1: Decidability results for Ad Hoc Networks and their sources

Result	Network type	Graphs	Specification	References
undecidability	AHN	All	COVER	[Delzanno et al. 2010]
undecidability	AHN	BD_k	COVER	[Delzanno et al. 2011]
undecidability for $k \geq 2$	AHN	BP_k	TARGET, REPEAT	[Delzanno et al. 2010]
undecidability	AHN	daAll	COVER	[Abdulla et al. 2013a]
decidability	AHN	BPC_k, BP_k, daBP_k	COVER	[Delzanno et al. 2010] [Delzanno et al. 2011] [Abdulla et al. 2013a]
decidability	LAHN	C	COVER, REPEAT, TARGET	[Delzanno et al. 2010]

7.7.1 VARIATIONS OF THE MODEL

Delzanno et al. [2012b] study AHNs with many other failure types (we considered only message losses). They distinguish between failures in nodes and in communication.

Node failures include non-deterministic restarts and crashes. They show by reduction from the undecidable COVER in AHNs (without failures) that the PMCPs for such models are undecidable.

Communication failures include non-deterministic message losses (covered here as LAHNS), message conflicts, and message conflicts with conflict detection. A message conflict happens when a node senses broadcasts from two or more nodes simultaneously. The model with message conflicts makes COVER decidable (and the proof uses the ideas from the LAHN section), while the model with conflict detection makes the PMCPs undecidable (via reduction from undecidable COVER for AHNs).

Abdulla et al. [2011] studied PMC problems for AHNs composed of processes being timed automata, and Delzanno et al. [2013] studied the case of LAHNS of processes being register automata.

Delzanno and Traverso [2013] studied PMC problems for AHNs where broadcasts are asynchronous. In this model each process has an unbounded (local) buffer for incoming messages. When a process broadcasts a message, it is enqueued into buffers of all receiving processes, and later the processes asynchronously handle the messages (in our model broadcasts are synchronous, i.e., a broadcast message is immediately and simultaneously handled by all receiving processes).

CHAPTER 8

Related Work

In this book we focused on established decidability results in parameterized model checking. As this topic is a lively field, we cannot make any claim about completeness of our survey. In particular, we left out the large body of research that includes invariant-based techniques, regular model checking, symbolic methods, techniques for counter automata, and abstraction-based methods. In this chapter we give a brief overview of these methods and give further references to the interested reader. Our classification of the techniques is unavoidably subjective, as except for several cornerstone papers, many techniques adopted multiple different ideas.

For the reader interested in practical applications of parameterized model checking techniques, we give an overview of PMC tools in Chapter 9.

8.1 ABSTRACTION TECHNIQUES

The general idea of abstraction-based techniques is as follows. One identifies a class of process templates that have a common structure.¹ To construct a new abstraction, one defines an abstraction mapping α that builds a finite-state system $\alpha(P)$ of a given process template P . Further, one proves that α preserves the specifications from a predefined class \mathcal{S} : for every specification $\varphi \in \mathcal{S}$ and every process template P , if $\alpha(P) \models \varphi$, then for every number of processes $n \geq 1$, it holds that $P^n \models \varphi$. This proof is the most complicated part of the technique. Typically, the proof is done by showing that $\alpha(P)$ simulates each system P^n , for $n \geq 1$, which implies that $\alpha(P)$ satisfies all ACTL^* -formulas that are satisfied by every system instance from P^n for $n \geq 1$ (see Clarke et al. [1999, Theorem 16]). Hence, for a given process template P , the parameterized model checking problem amounts to constructing the abstraction $\alpha(P)$ and checking whether the property holds for $\alpha(P)$.

Abstraction introduces new behavior and thus, when $\alpha(P)$ does not satisfy the property φ to be verified, one cannot immediately conclude whether the property is satisfied in the parameterized family $\{P^n \mid n \geq 1\}$. If $\alpha(P) \not\models \varphi$, the model checking tool will provide a counterexample, and the user has to analyze the counterexample with the goal of refining the abstraction $\alpha(P)$. Sometimes, $\alpha(P)$ can be refined automatically, see e.g., Abdulla et al. [2010] and John et al. [2013]. However, refinement techniques for parameterized model checking are, by design, incomplete. Therefore, abstraction-based techniques do not characterize decidable classes of the parameterized model checking problem, in contrast to the techniques described in this book.

¹For instance, $\{0, 1, \infty\}$ -counter abstraction by Pnueli et al. [2002] is applicable to process templates that can be represented with the SLP language.

130 8. RELATED WORK

Notwithstanding, this incompleteness property, abstraction methods have been successfully applied to many practical examples.

Pnueli et al. [2002] highlighted $\{0, 1, \infty\}$ -counter abstraction. There, a state of the abstraction $\alpha(P)$ consists of global variables over some finite domain and abstract counters $\kappa_1, \dots, \kappa_m$ over the domain $\{0, 1, 2\}$. Counter κ_i represents the abstract number of processes in the i -th local state as follows: value 0 corresponds to zero processes; value 1 corresponds to exactly one process; value 2 corresponds to “many” processes (i.e., more than one). By using the finite abstract domain to count processes, one constructs the finite abstraction $\alpha(P)$ for a process template P . The atomic propositions allow one to check how many processes there are in a certain local state, e.g., $\kappa_{\ell_1} = 1$ asserts that there is exactly one process in the local state ℓ_1 , while $\kappa_{\ell_2} > 1$ asserts that several processes are in the state ℓ_2 . Pnueli et al. [2002] showed that for every 1-LTL\X-formula φ , if $\alpha(P) \models \varphi$ holds, then $\forall n \geq 1. P^n \models \varphi$ holds as well. It is easy to see that this abstraction works especially well for mutual exclusion properties: the formula $G(\kappa_{\ell_C} < 2)$ expresses that never two processes are in the critical section at the same time. Pnueli et al. [2002] applied their abstraction to reason about safety and liveness of Szymanski’s mutual exclusion algorithm and of simplified Bakery. The abstractions were checked with the NuSMV model checker [Cimatti et al., 2002].

Ip and Dill [1999] discussed an abstraction similar to $\{0, 1, \infty\}$ -counter abstraction, but they focused only on safety properties. They implemented their abstraction in Murφ [Dill, 1996, 2008] and verified safety of Peterson’s mutual exclusion algorithm and of the cache coherence protocols MESI, ICCP, and DCCP.

John et al. [2013] generalized $\{0, 1, \infty\}$ -counter abstraction to counter abstraction over *parametric intervals* (PIA-counter abstraction). In their abstraction, each counter is assigned an abstract value that corresponds to an interval, e.g., if $\kappa_i = [(n+1)/2, n+1]$, then a majority out of n processes is in state ℓ_i . Hence, $\{0, 1, \infty\}$ -counter abstraction is a special case of PIA-counter abstraction over the intervals $[0, 1]$, $[1, 2]$, and $[2, n+1]$. John et al. [2013] implemented their abstraction in the tool ByMC and verified safety and liveness of reliable broadcast algorithms in the presence of a parameterized number of faults (e.g., crash and Byzantine). The abstract systems were checked with the Spin model checker [Holzmann, 2003].

Clarke et al. [2008] introduced environment abstraction, which can also be seen as a generalization of $\{0, 1, \infty\}$ -counter abstraction. In their framework, an abstraction of a global state is a formula that describes the state of a special process (called the *reference* process) and a summary of the states of the other processes (called the *environment*). A system transition is abstracted as a step of the reference process as follows: the reference process checks whether the environment satisfies the precondition; if so, the reference process changes its local state and the state of the environment. After the transition has been made, another process in the environment can become the reference process. The abstract semantics of the transitions is defined individually for each protocol class. Clarke et al. [2008] applied their technique to verify the following protocols: readers/writers, Szymanski’s mutual exclusion, simplified Bakery, German’s cache coherence pro-

tocol, and the Flash cache coherence protocol. Their work was extended by Talupur and Tuttle [2008].

Basler et al. [2009] applied $\{0, 1, \infty\}$ -counter abstraction on boolean programs to verify concurrent programs with the parameterized number of threads. They proposed a symbolic exploration algorithm that has been implemented in the tool BOOM and scales well for boolean programs in practice.

Jensen and Lynch [1998] introduced a parameterized simulation relation and used it to prove soundness of their specific abstraction of Burns' mutual exclusion algorithm. In their abstraction, they mapped a system of n processes to a system of two processes AP_0 and AP_1 : intuitively, AP_0 and AP_1 , respectively, represent the processes with indices i and j for $1 \leq i < j \leq n$. Jensen and Lynch [1998] proved soundness of the abstraction in the Larch Proof Assistant and verified safety of the abstraction with Spin.

Calder and Miller [2003] verified the Firewire protocol with Spin and introduced a specific abstraction for the parameterized case.

8.2 REGULAR MODEL CHECKING

Regular model checking (RMC) is a symbolic technique built upon automata representation of transition relations; see Abdulla [2012] for a detailed survey. RMC assumes a linear order \prec on a set of processes. Such an order comes naturally when the processes of a parameterized system are organized in a ring or a line. Let P be a process template with a *finite* set Q of local states. Then, a global state of a system P^n is represented as a word $w \in Q^n$ of length n , where the letter $w[i]$ corresponds to the local state of the i -th process (in the order \prec) for $1 \leq i \leq n$.

In RMC, the set of global states S is required to be a language of a finite-state automaton A_S over the alphabet Q . Similarly, a finite-state automaton A_R over the alphabet $Q \times Q$ represents the union of transition relations in the parameterized family $\{P^n \mid n \geq 1\}$. As the automaton A_R models systems that do not dynamically create or destroy processes, the language of A_R is size-preserving, that is, the language $\mathcal{L}(A_R)$ of the automaton A_R is a subset of $\{(w, w') \mid w, w' \in Q, |w| = |w'|\}$.

Given a process template P , one has to construct the following automata that represent the union of the transition systems in the parameterized family $\{P^n \mid n \geq 1\}$:

- an automaton A_I to represent the set of the initial states,
- an automaton A_B to represent the set of bad states (violating the reachability specification), and
- an automaton A_R to represent the transition relation.

These automata are given as the input to the RMC technique. Their construction is not part of the RMC technique.

The main goal of the RMC technique is to compute the reflexive and transitive closure A_R^* of the automaton A_R : an automaton that only accepts pairs of words $w, w' \in Q^*$ that are connected by a path $w_1, \dots, w_k \in Q^*$, that is, $w = w_1$, $w_k = w'$ and $(w_i, w_{i+1}) \in \mathcal{L}(A_R)$ for $1 \leq i < k$. Once A_R^* is built, one applies automata-theoretic operations to check whether one of the systems in the family $\{P^n \mid n \geq 1\}$ violates the specification: let us denote by $\mathcal{L}(A_R^*(A_I))$ the set of words $\{w' \in Q^* \mid w \in \mathcal{L}(A_I), (w, w') \in \mathcal{L}(A_R^*)\}$, which symbolically represents all reachable states. Then, no system P^n reaches a bad global state (i.e., a state in the language of the automaton A_B), if and only if $\mathcal{L}(A_R^*(A_I)) \cap \mathcal{L}(A_B) = \emptyset$.

Computing A_R^* is the main bottleneck of the RMC technique. Indeed, in general, the closure is neither regular, nor computable. Several less general solutions were introduced, e.g., *acceleration* of process transitions [Abdulla et al., 1999], *quotienting* of automata by state merging [Abdulla et al., 2002, Bouajjani et al., 2000], and *abstraction* [Bouajjani et al., 2004].

We make two observations about the representation. First, alphabet Q is finite and for each $n \geq 1$, automaton A_S accepts finitely many words of length n . However, as the language of A_S can be infinite, the automaton is able to capture global states of all systems P^n , for $n \geq 1$. This brings us to the second observation: not every set of global states S is regular, i.e., there may not be an ordering \prec under which S can be encoded with a finite-state automaton, and thus RMC techniques may not apply. The same applies to the automaton A_R .

Finally, To and Libkin [2010] introduced algorithmic metatheorems that generalize reasoning found in regular model checking research. A tool that implements regular model model checking is T(O)RMC [Legay, 2008].

8.3 SYMBOLIC TECHNIQUES

Similar to regular model checking (Section 8.2), symbolic techniques do not represent state sets and transition relations explicitly, but use symbolic representations. While RMC is based on one form of symbolic representation, namely, finite automata, the symbolic techniques presented in this section apply various logical theories to represent states and transitions.

Henriksen et al. [1995] proposed to use *monadic second order logic* as a “highly succinct alternative to regular expressions.” Elgaard et al. [1998] implemented their framework in the tool MONA.

General principles for parameterized symbolic model checking were laid out by Kesten et al. [1997]. There, the authors used a *finitary second-order theory of one successor* (FS1S) and FS* S , which have the same expressive power as finite automata and tree automata respectively. Further, Pnueli and Shahar [2000] investigated liveness and acceleration in this framework. These techniques were implemented in the tool called TLV [Pnueli and Shahar, 1996], and have been re-implemented in JTLV [Pnueli et al., 2010].

Maidl [2001] introduced a framework, in which parameterized systems are represented as formulas over Presburger arithmetic. Apart from local and global variables, the formulas contain special variables that represent process indices. The framework encompasses token passing and

broadcast systems. The authors give a semi-algorithm based on proof tree construction and give sufficient conditions of its termination for safety properties.

Reachability checking for array-based systems. There are a number of techniques that are similar to RMC in that they model the state of a parameterized system as a finite word, or equivalently an array of values from some logical background theory, but are different from RMC in that they represent sets of states and the transition relation of the system as constraints in this theory.

A first representative of this approach developed directly out of RMC and was initially called *regular model checking without transducers* [Abdulla et al., 2007]. The system model assumes an order on the processes, and allows them to communicate via shared variables, rendezvous, or broadcast. In addition, transitions can be guarded with global conditions that take into account the order of processes. Systems are verified by an approximate backward reachability check, where potentially infinite sets of system configurations are represented as (finite) constraints on system variables. To ensure termination, the approach considers a *monotonic abstraction* of the transition relation, which guarantees that transitions are monotonic with respect to the subword relation on configurations. As a result, the algorithm can always work on *upwards closed sets* of configurations, and thus one considers an abstraction of the original system which is a *well-structured transition system*. The approach has later been extended by Abdulla et al. [2009] to systems with infinite-state components by also supporting numerical variables. To ensure decidability of the reachability computation, sets of valuations of numerical variables can only be expressed as *gap-order constraints*. The tool Undip implements both the finite- and infinite-state case. As an alternative to monotonic abstraction, the approach for finite-state processes has recently been combined with counter abstraction, which in contrast to monotonic abstraction allows to refine the abstraction if verification is not successful [Ganty and Rezine, 2014]. An implementation of this extension is available in the tool PWC.

Ghilardi et al. [2008] introduced an approach that generalizes the basic idea of representing states and transitions as constraints on arrays over a suitable background theory. They define the notion of *array-based systems*, where a state of the system is represented as an unbounded array of values from a (parametric) first-order theory of elements. As another generalization, the approach also supports a parametric index theory for the array, allowing to model non-linear topologies. Then, sets of states can be represented as formulas over the theories of indices and elements, and transition relations as formulas that relate the possible pre- and post-states. In both cases, formulas can be quantified, making the problem potentially undecidable due to the undecidability of first-order logic. Instead of an abstraction to a well-structured system, this approach uses SMT solvers and quantifier elimination methods to directly reason on these constraints. For combinations of systems and theories such that all quantifiers can be effectively eliminated and the ground satisfiability problem is decidable, one obtains the fully automated technique of *Model Checking Modulo Theories (MCMT)* [Ghilardi and Ranise, 2010]. In addition to an implementation (called MCMT) by its inventors, the approach has been implemented by Conchon et al. [2012] in the tool Cubicle.

Completeness of MCMT relies on the one hand on existing decidability results for first-order theories, and on the other hand on proving new results for decidability or the possibility of quantifier elimination in the theories that are necessary for describing systems and properties. For some expressive classes of array-based systems, full model checking may not be possible. In such cases, one may resort to invariant checking or bounded model checking, which can both be expressed as a single first-order constraint over a suitable theory. The framework of *local theory extensions* allows to define configurations and transitions of array-based systems as extensions of a given first-order theory, providing a decision procedure for such constraints [Faber et al., 2010, Jacobs and Sofronie-Stokkermans, 2007].

8.4 DYNAMIC CUTOFF DETECTION

Many of the results presented in this survey solve the PMCP by providing a *cutoff*, i.e., a maximal number of processes that need to be considered for a given class of systems and specifications. While such cutoff results let us decide the PMCP, they are restricted to the classes of systems and specifications for which the proof methods have been designed. A related approach is *dynamic cutoff detection*, which does not depend on *a priori* proven cutoff theorems, but instead tries to automatically detect a suitable cutoff for a given system template, and possibly also a given specification.

Kaiser et al. [2010] considered the problem of finding cutoffs that are specific to a given parameterized system and a safety property. To this end, their algorithm first computes the set of reachable *thread-states*, i.e., combinations of shared and local state, in a system of fixed size k . For the class of shared-memory systems under consideration, a sufficient and necessary condition for the existence of new reachable thread-states in systems of size $n > k$ is identified. In a nutshell, if any new thread-states are reachable for $n > k$, then among those there is one thread-state (s, l) with minimal distance from the initial state. Furthermore, by the characteristics of their model, this thread-state (s, l) must be reachable from those that are reached in a system of size k by a transition that does not change the shared memory part s . This gives rise to an analysis for *thread-state candidates* that could be reached in a system of size $k + 1$. Whether this actually is the case can be decided for each candidate by a backward coverability analysis, which is decidable for fixed k . Since there can only be a finite number of candidates, this procedure will terminate for the given k . If none of the candidates can be reached, then k is a cutoff, otherwise the procedure is iterated for $k + 1$.

Abdulla et al. [2013b] also detected property- and system-specific cutoffs for safety properties. Their system model is close to the one presented in Chapter 6, with connectivity graphs that can be different from cliques and guarded updates that can take into account this topology. To find a cutoff, the procedure computes in parallel the exact set of reachable states for a system of size k , and an over-approximation of the reachable states with respect to a *view abstraction*, also parameterized by k . Roughly, for this overapproximation we view a global state as a word over local states of the system, and not only consider words that are reachable, but also words that

can be constructed from subwords of reachable words. By a separate argument, the authors show that for their class of systems it is sufficient to consider words of size $k + 1$ in this procedure in order to obtain completeness. If the concrete computation finds an error trace, then the given safety property does not hold. If the abstract computation does not find an error trace, then k is a cutoff and the safety property holds for all instances of the system. If neither is the case, then the procedure is iterated for $k + 1$.

The dynamic cutoff techniques are applicable to very expressive classes of systems, such that the PMCP is in general undecidable even for safety properties. Thus, the search for a suitable cutoff can in general not be guaranteed to terminate, but only in restricted cases that are decidable. For instance, the approach by [Abdulla et al. \[2013b\]](#) is guaranteed to terminate for well-quasi ordered transition systems. Cutoffs that are specific to a property and a process template can be much smaller than pre-determined cutoffs that hold for *all* elements of a class of systems and properties.

8.5 NETWORK INVARIANTS

Invariant-based techniques use inductive arguments to reason about parameterized systems. Intuitively, these arguments state that if a certain property is satisfied in a system of n processes it also holds in a system of $n + 1$ processes. There is a number of such techniques in the literature that apply induction to reason about parameterized systems. The techniques discussed here thus exploit invariants over a sequence of systems, rather than invariants of computations, that is, over a sequence of steps. The techniques discussed differ in the following ingredients.

(invariant specification) This defines the object that is invariant for a sequence of system instances, e.g., an invariant can be a labeled transitions system that simulates every system instance in the sequence.

(invariant preservation) A formal definition of what it means for a system instance P^n to preserve an invariant, e.g., simulation.

(inductive step) An inductive step to prove that a system instance P^{n+1} preserves an invariant, if a system instance P^n does.

The first results on using induction for parameterized model checking were obtained independently by [Kurshan and McMillan \[1989\]](#) and [Wolper and Lovinfosse \[1989\]](#). [Wolper and Lovinfosse \[1989\]](#) introduced the framework of *network invariants*. [Kurshan and McMillan \[1989\]](#) introduced the *structural induction theorem*.

In the framework of network invariants, an invariant I is specified in the same language as a process template P , namely, as a process in the framework of Hoare's Communicating Sequential Processes (CSP). Invariant preservation is defined with the means of an implementation relation \preceq on labeled transition systems, e.g., trace inclusion, bisimulation, or observational equivalence. The technique consists of proving two properties for some $k \geq 1$:

136 8. RELATED WORK

- the base system P^k implements I , i.e., $P^k \preceq I$, and
- the parallel composition of I and P implements I , i.e., $I \parallel P \preceq I$.

[Wolper and Lovinfosse \[1989\]](#) showed by induction that whenever P and I satisfy these two properties, it holds that $P^n \preceq I$ for $n \geq k$. Hence to prove a property for all n , it just remains to check the specification against I , and against instances P^1, \dots, P^{k-1} . The crucial point of this method is how to generate invariants. One may try $I = P$ or $I = P^2$, which works for simple examples. If this guess fails, an invariant candidate must be provided by the user.

[Kurshan and McMillan \[1989\]](#) obtained similar results in the form of the structural induction theorem in the framework of Milner's Calculus of Communicating Systems (CCS). They also note that one can use various implementation relations, e.g., “may” preorder on CCS processes, language containment, strong and weak bisimulation, or the “implements” relation of I/O automata.

Network invariants and network grammars. [Shtadler and Grumberg \[1990\]](#) extended the framework of network invariants to parameterized systems whose connectivity graph is derived by a network grammar. In their work, they used a special form of computation equivalence.

[Clarke et al. \[1995\]](#) also used network grammars to capture the connectivity graph. Instead of equivalence relations, e.g., bisimulation, they use a preorder on labeled transition systems, namely, simulation. The key feature of their work is to specify expected behavior with automata over regular expressions,² as opposite to indexed temporal logics. For invariant generation, [Clarke et al. \[1995\]](#) gave a method to generate invariant candidates. The method takes the specification as input and has better chances of generating useful invariants than a more general technique that does not consider the specification. Still, if the generated invariant candidate is not an invariant, the user must help the verification tool. The original framework was formulated for synchronous systems. Later, [Clarke et al. \[1997\]](#) extended the framework to asynchronous systems.

[Lesens et al. \[1997\]](#) followed up the work by [Clarke et al. \[1995\]](#) and introduced several invariant synthesis heuristics based on *widening techniques*. If the technique fails to find an invariant, the user can manually refine the widening operator and thus give a hint to the technique. This is the key improvement over other techniques, which cannot be tuned by the user. [Kesten and Pnueli \[2000\]](#) transferred the framework of network invariants to fair-discrete systems, in which processes communicate via shared variables, as opposed to rendezvous communication used in the other work discussed in this section. [Konnov and Zakharov \[2010\]](#) also applied network grammars to generate connectivity graphs. As simulation is typically too restrictive for asynchronous systems, they introduced several weak forms of simulation: block simulation, quasi-block simulation, and semi-block simulation. They have implemented the technique in the tool CHEAPS.

²Similar to regular model checking, the alphabet is the set of local states \mathcal{Q} .

8.6 INVISIBLE INVARIANTS

The notion of invariants used in the techniques based on invisible invariants is closer to the one used in the analysis of programs (loop invariants) or algorithms. Pnueli et al. [2001] introduced an approach to invariant detection in concurrent programs parameterized by the number of processes. The method targets verification of k -indexed formulas, e.g., the 2-indexed formula $\forall i, j \neq i. G(at_i \neq critical \vee at_j \neq critical)$. In the technique of invisible invariants, a model checker is run on a fixed-size system instance to collect the reachable *global* states R . Given k , the technique projects the reachable global states R on all combinations of k process indices and thus computes the reachable combinations of the *local* states of k processes. The result of this projection—that is, the set of combinations of local states—is called an *invisible invariant*. For parametrized model checking, Pnueli et al. [2001] proved a small model property for concurrent programs with (bounded) shared variables, from which follows that an invisible invariant of a fixed-size system instance is also an invariant of the parameterized system.

Fang et al. [2006] combined invisible invariants with ranking functions, in order to verify liveness properties. McMillan and Zuck [2011] investigated connections between invisible invariants and abstract interpretation. Johnson and Mitra [2012] extended the small model theorem by Pnueli et al. [2001] to the networks of rectangular hybrid automata. They implemented their technique in the tool PASSEL [Johnson, 2013].

8.7 OTHER ASPIRING APPROACHES

Fully symmetric parameterized systems, e.g., cache coherence protocols, can be naturally encoded as *counter automata* [Delzanno, 2003, Leroux and Sutre, 2005]. A counter automaton is a graph, whose edges are labeled with counter updates and guards over counters and parameters (such as the number of processes). Typically, the guards and updates are expressed in linear integer arithmetic. Delzanno [2003] applied backward reachability on the systems with linear arithmetic constraints to analyze cache coherence protocols. Leroux and Sutre [2005] introduced flat (and flatable) counter automata, whose reachability can be checked with terminating acceleration techniques. This technique is implemented in the tool FAST [Bardin et al., 2008].

Esparza et al. [2013] introduced *non-atomic networks* to model parameterized asynchronous shared memory systems. A non-atomic network consists of a single leader and many contributors that write to and read from a shared register store. The authors showed that safety verification of non-atomic networks of processes modeled with state machines is decidable and at least PSPACE-hard. Further, they showed that safety verification of non-atomic networks of processes modeled with pushdown machines is undecidable.

Bouajjani et al. [2008] studied parameterized model checking of *resource allocation systems* (RASs). Such systems have a bounded number of resources, each owned by at most one process at any time. Processes are pushdown automata, and can request resources with high or normal priority. RASs are similar to conjunctive guarded protocols in that certain transitions are disabled

138 8. RELATED WORK

unless a process has a certain resource. RASs without priorities and with processes being finite state automata can be converted to conjunctive guarded protocols (at the price of blow up), but not vice versa. The authors study parameterized model checking wrt. $LTL \setminus X$ properties on strong-fair or all runs, and wrt. (local or global) deadlocks on all runs. The proof structure resembles that of [Emerson and Kahlon \[2000\]](#).

[Farzan et al. \[2014\]](#) introduced *counting proofs* to reason about parameterized shared memory programs. Their technique automatically generates auxilliary counters and synthesizes a small proof of partial correctness of a concurrent program. [Farzan et al. \[2015b\]](#) also introduced *proof spaces*, where they construct a finite set of correctness proofs that capture all traces of a parameterized program. The proofs are constructed from predicate automata [[Farzan et al., 2015a](#)], an infinite-state generalization of alternating finite automata.

CHAPTER 9

Parameterized Model Checking Tools

The results covered in this book show that the parameterized model checking problem is only decidable for rather restricted classes of system models and specifications. In addition to that, parameterized model checking is computationally hard even in cases where it is decidable. As a result, there are only few software tools that implement decision procedures for the PMCP, and most of the available tools are implementations of the semi-decision procedures covered in Chapter 8. For researchers who want to get some hands-on experience with parameterized model checking, we give an overview of parameterized model checking tools that are available at the time of this writing. Table 9.1 summarizes the approaches of the available tools, and Table 9.2 shows where they can be obtained.

The only tool that uses decidability results from this book is PARTY [Khalimov et al., 2013b]. PARTY supports synthesis of concurrent systems in a token ring topology, with specifications in indexed LTL\X. It uses the cutoff results from Chapter 4, together with an SMT-based approach to solve the synthesis problem for a process template that satisfies the specification in the cutoff topology. By constraining the implementation of the process template such that it only allows one implementation, the approach can also be used to solve the PMCP in token rings.

Several of the approaches from the previous chapter have been implemented in tools such as BOOM [Basler et al., 2009], T(O)RMC [Legay, 2008], MONA [Elgaard et al., 1998], TLV [Pnueli and Shahar, 1996] and JTLV [Pnueli et al., 2010], Undip [Abdulla et al., 2009], MCMT [Ghilardi and Ranise, 2010], CHEAPS [Konnov and Zakharov, 2010], Cubicle [Conchon et al., 2012], ByMC [John et al., 2013], and PCW [Ganty and Rezine, 2014]. We refer to the previous chapter for an explanation of their respective approaches to parameterized model checking.

Furthermore, there is a number of tools that implement more general forms of infinite-state model checking, in particular with support for integer-valued variables. With a suitable encoding of parameterized systems into systems with integer variables, these tools can be used for parameterized model checking. Examples of such tools are ALV [Yavuz-Kahveci and Bultan, 2009], BRAIN [Rybina and Voronkov, 2002], FAST [Bardin et al., 2008], and nuXmv [Cavada et al., 2014].

Finally, there is a number of tools for the verification of hybrid systems, like KeYmaera [Platzer, 2012, Platzer and Quesel, 2008] and Passel [Johnson, 2013]. Both systems allow

140 9. PARAMETERIZED MODEL CHECKING TOOLS

Table 9.1: Parameterized model checking tools: approaches

Tool	Authors	Implements
ALV	Yavuz-Kahveci and Bultan [2009]	approximate infinite-state reachability
BRAIN	Rybina and Voronkov [2002]	symbolic infinite-state reachability
BOOM	Basler et al. [2009]	counter abstraction
ByMC	John et al. [2013]	counter abstraction
CHEAPS	Konnov and Zakharov [2010]	network invariants
Cubicle	Conchon et al. [2012]	symbolic backward reachability
FAST	Bardin et al. [2008]	acceleration-based inf.-state reachability
IIV	Balaban et al. [2005]	invisible invariants
JTLV	Pnueli et al. [2010]	semi-automatic verification platform
KeYmaera	Platzer and Quesel [2008]	semi-automatic verification platform
Murφ	Dill [1996]	counter abstraction
MCMT	Ghilardi and Ranise [2010]	symbolic backward reachability
MONA	Henriksen et al. [1995]	monadic second-order logic
NUXMV	Cavada et al. [2014]	infinite-state model checking
PARTY	Khalimov et al. [2013b]	(static) cutoff detection
Passel	Johnson [2013]	invisible invariants
PCW	Ganty and Rezine [2014]	symbolic reachability, counter abstraction
TLV	Pnueli and Shahar [1996]	semi-automatic verification platform
T(O)RMC	Legay [2008]	regular model checking
Undip	Abdulla et al. [2009]	approximate backward reachability

for the verification of distributed hybrid systems with a parametric number of components. While Passel is based on a completely automatic combination of reachability checking and detection of invisible invariants, KeYmaera is a very powerful semi-automatic theorem prover.

Table 9.2: Parameterized model checking tools: availability

Tool	URL
ALV	http://www.cs.ucsb.edu/~bultan/composite/
BOOM	http://www.cprover.org/boom/
BRAIN	http://www.cs.man.ac.uk/~voronkov/BRAIN/
ByMC	http://forsyte.tuwien.ac.at/software/bymc/
CHEAPS	http://lvk.cs.msu.ru/~konnov/cheaps/index_en.html
Cubicle	http://cubicle.lri.fr
FAST	http://tapas.labri.fr/trac/wiki/FASTER
JTLV	http://jtlv.yaar.net/
KeYmaera	http://symbolaris.com/info/KeYmaera.html
MCMT	http://users.mat.unimi.it/users/ghilardi/mcmt/
MONA	http://www.brics.dk/mona/
Murφ	http://formalverification.cs.utah.edu/
NUXMV	Murphi/https://nuxmv.fbk.eu
PARTY	https://github.com/5nizza/Party
Passel	https://publish.illinois.edu/passel-tool/
PWC	http://www.ahmedrezine.com/tools/
TLV	http://www.cs.nyu.edu/acsys/tlv/
T(O)RMC	http://www.montefiore.ulg.ac.be/~legay/TORMC/index-tormc.html
Undip	http://www.ahmedrezine.com/tools/

CHAPTER 10

Conclusions

In this book, we provided a general model for uniform concurrent systems that captures a large class of systems from the literature. Our model includes different forms of communication, like token-passing, rendezvous, or broadcast, as well as different communication graphs, like cliques, rings, stars, or even dynamic topologies that change at runtime.

For this class of systems, we surveyed the existing decidability results, drawing a map that focuses on the border between decidability and undecidability, and highlighting some of the uncharted territory in this research area.

Regarding undecidability results, we noticed that unrestricted forms of communication make it possible even for uniform parameterized systems to simulate Turing machines. Such communication is thus a source of undecidability. This even holds for quite simple systems, e.g., with a simple token in a bidirectional ring architecture, or a two-valued token in a unidirectional ring architecture. The reason is essentially that arbitrary amounts of information can be exchanged over an infinite run of a system, even if a single synchronization can only transmit one bit of information. Typically, this allows to represent Turing machines with tapes of arbitrary length, which leads to undecidability.

Many (but not all) decidability results give a constructive way to solve the PMCP for interesting subclasses, e.g., by a decomposition into several finite-state model checking problems. However, in many cases there either is no constructive result, or the decomposition is too big to be model checked in practice. Thus, additional research is needed to obtain effective ways to solve the PMCP.

Regarding specifications, it is worth noting that the token passing systems of Chapter 4 are the only ones known to us for which branching-time logics, like certain fragments of indexed- $\text{CTL}^*\backslash X$, are known to have decidable PMCP. This is in contrast to pairwise rendezvous and disjunctively guarded systems that are known to have undecidable PMCP for CTL^* , while for ad hoc networks and conjunctively guarded systems the problem does not seem to have been studied.

Another set of open problems concerns lower bounds and computational complexity of the PMCP. A few exact bounds for PMCP are known: rendezvous systems and 1-index $\text{LTL}\backslash X$ specifications are EXPSPACE-complete (and PSPACE-complete if there is no controller) [Esparza, 2014, German and Sistla, 1992]; disjunctively guarded systems and 1-index $\text{LTL}\backslash X$ specifications are PSPACE-complete [Aminof et al., 2014b]; and broadcast systems and safety specifications have “Ackermanian complexity” [Schmitz and Schnoebelen, 2013].

144 10. CONCLUSIONS

While the theoretical results presented give some insight in the problems of verifying large concurrent systems, it should be noted that the systems considered here are still quite limited compared to practical concurrent programs. In this book, we considered parallel compositions $P \parallel \dots \parallel P$ of n copies of a fixed finite-state process P — that is, P is independent of n . As systems consist of copies of the same process, processes cannot use unique identifiers (IDs). IDs are, however, quite natural in many applications, and from a theoretical viewpoint they are quite powerful as they can be used to break symmetries. Similarly, the processes have a fixed state space independent of the system size. Consequently, processes cannot use counting arguments (e.g., a majority of processes are in certain states) that are useful to decide on coordinated actions.

Of course, we can think of more complicated systems. Consider parallel compositions $P(1, n) \parallel \dots \parallel P(n, n)$ where $P(i, n)$ is a finite-state process that has a unique ID i as well as “knowledge” of the number of processes n in the system; consequently, the local state space of the processes is parameterized. Such systems seem quite natural, and actually there is a sizeable amount of literature on such systems, e.g., in the area of distributed algorithms [[Attiya and Welch, 2004](#), [Lynch, 1996](#)]. This model is rarely addressed in the parameterized model checking literature, and as discussed above, in most cases the PMCP is undecidable. However, we believe that in this area theoretical work must also be done in order to design procedures that allow one to reason about such systems.

Bibliography

Parosh Aziz Abdulla. Regular model checking. *STTT*, 14(2):109–118, 2012. DOI: [10.1007/s10009-011-0216-8](https://doi.org/10.1007/s10009-011-0216-8). 131

Parosh Aziz Abdulla, Kārlis Čerāns, Bengt Jonsson, and Yih-Kuen Tsay. General decidability theorems for infinite-state systems. In *LICS*, pages 313 –321, 1996. DOI: [10.1109/LICS.1996.561359](https://doi.org/10.1109/LICS.1996.561359). 25, 26, 99

Parosh Aziz Abdulla, Ahmed Bouajjani, Bengt Jonsson, and Marcus Nilsson. Handling global conditions in parametrized system verification. In *CAV*, pages 134–145. Springer, 1999. DOI: [10.1007/3-540-48683-6_14](https://doi.org/10.1007/3-540-48683-6_14). 132

Parosh Aziz Abdulla, Bengt Jonsson, Marcus Nilsson, and Julien d’Orso. Regular model checking made simple and efficient. In *CONCUR*, pages 116–130, 2002. DOI: [10.1007/3-540-45694-5_94](https://doi.org/10.1007/3-540-45694-5_94). 132

Parosh Aziz Abdulla, Giorgio Delzanno, Noomene Ben Henda, and Ahmed Rezine. Regular model checking without transducers (on efficient verification of parameterized systems). In *TACAS*, volume 4424 of *LNCS*, pages 721–736. Springer, 2007. DOI: [10.1007/978-3-540-71209-1_56](https://doi.org/10.1007/978-3-540-71209-1_56). 133

Parosh Aziz Abdulla, Giorgio Delzanno, and Ahmed Rezine. Approximated parameterized verification of infinite-state processes with global conditions. *Formal Methods in System Design*, 34(2):126–156, 2009. DOI: [10.1007/s10703-008-0062-9](https://doi.org/10.1007/s10703-008-0062-9). 133, 139

Parosh Aziz Abdulla, Yu-Fang Chen, Giorgio Delzanno, Frédéric Haziza, Chih-Duo Hong, and Ahmed Rezine. Constrained monotonic abstraction: A CEGAR for parameterized verification. In *CONCUR*, pages 86–101, 2010. DOI: [10.1007/978-3-642-15375-4_7](https://doi.org/10.1007/978-3-642-15375-4_7). 129

Parosh Aziz Abdulla, Giorgio Delzanno, Othmane Rezine, Arnaud Sangnier, and Riccardo Traverso. On the verification of timed ad hoc networks. In *FORMATS*, volume 6919 of *LNCS*, pages 256–270. Springer, 2011. DOI: [10.1007/978-3-642-24310-3_18](https://doi.org/10.1007/978-3-642-24310-3_18). 128

Parosh Aziz Abdulla, Mohamed Faouzi Atig, and Othmane Rezine. Verification of directed acyclic ad hoc networks. In *FORTE*, volume 7892 of *LNCS*, pages 193–208. Springer, 2013a. DOI: [10.1007/978-3-642-38592-6_14](https://doi.org/10.1007/978-3-642-38592-6_14). 19, 103, 107, 115, 116, 117, 118, 119, 120, 122

146 BIBLIOGRAPHY

- Parosh Aziz Abdulla, Frédéric Haziza, and Lukáš Holík. All for the price of few. In *VMCAI*, volume 7737 of *LNCS*, pages 476–495. Springer, 2013b. DOI: [10.1007/978-3-642-35873-9_28](https://doi.org/10.1007/978-3-642-35873-9_28). 134, 135
- Benjamin Aminof, Swen Jacobs, Ayrat Khalimov, and Sasha Rubin. Parameterized model checking of token-passing systems. In *VMCAI*, volume 8318 of *LNCS*, pages 262–281, January 2014a. DOI: [10.1007/978-3-642-54013-4_15](https://doi.org/10.1007/978-3-642-54013-4_15). 31, 35, 39, 40, 41, 42, 45, 46, 47
- Benjamin Aminof, Tomer Kotek, Sasha Rubin, Francesco Spegni, and Helmut Veith. Parameterized model checking of rendezvous systems. In *CONCUR*, volume 8704, pages 109–124. Springer, 2014b. DOI: [10.1007/978-3-662-44584-6_9](https://doi.org/10.1007/978-3-662-44584-6_9). 46, 63, 101, 143
- Benjamin Aminof, Sasha Rubin, Florian Zuleger, and Francesco Spegni. Liveness of parameterized timed networks. In *ICALP (Part II)*, volume 9135 of *LNCS*, pages 375–387, 2015. DOI: [10.1007/978-3-662-47666-6_30](https://doi.org/10.1007/978-3-662-47666-6_30). 64
- Krysztof R. Apt and Dexter C. Kozen. Limits for automatic verification of finite-state concurrent systems. *Information Processing Letters*, 15:307–309, 1986. DOI: [10.1016/0020-0190\(86\)90071-2](https://doi.org/10.1016/0020-0190(86)90071-2). 2, 3
- Hagit Attiya and Jennifer Welch. *Distributed Computing*, 2nd ed. John Wiley & Sons, 2004. DOI: [10.1002/0471478210](https://doi.org/10.1002/0471478210). 144
- Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT Press, 2008. 1, 2, 12, 14, 83, 97
- Ittai Balaban, Yi Fang, Amir Pnueli, and Lenore D. Zuck. IIV: an invisible invariant verifier. In *CAV*, pages 408–412, 2005.
- Valmir C. Barbosa and Eli Gafni. Concurrency in heavily loaded neighborhood-constrained systems. *ACM Trans. Program. Lang. Syst.*, 11(4):562–584, 1989. DOI: [10.1145/69558.69560](https://doi.org/10.1145/69558.69560). 48
- Sébastien Bardin, Alain Finkel, Jérôme Leroux, and Laure Petrucci. FAST: acceleration from theory to practice. *STTT*, 10(5):401–424, 2008. DOI: [10.1007/s10009-008-0064-3](https://doi.org/10.1007/s10009-008-0064-3). 137, 139
- Gerard Basler, Michele Mazzucchi, Thomas Wahl, and Daniel Kroening. Symbolic counter abstraction for concurrent software. In *CAV*, volume 5643 of *LNCS*, pages 64–78. Springer, 2009. DOI: [10.1007/978-3-642-02658-4_9](https://doi.org/10.1007/978-3-642-02658-4_9). 131, 139
- Ahmed Bouajjani, Bengt Jonsson, Marcus Nilsson, and Tayssir Touili. Regular model checking. In *CAV*, pages 403–418, 2000. DOI: [10.1007/10722167_31](https://doi.org/10.1007/10722167_31). 132

BIBLIOGRAPHY 147

- Ahmed Bouajjani, Peter Habermehl, and Tomás Vojnar. Abstract regular model checking. In *CAV*, pages 372–386, 2004. DOI: [10.1007/978-3-540-27813-9_29](https://doi.org/10.1007/978-3-540-27813-9_29). 132
- Ahmed Bouajjani, Peter Habermehl, and Tomás Vojnar. Verification of parametric concurrent systems with prioritised FIFO resource management. *Formal Methods in System Design*, 32(2):129–172, 2008. DOI: [10.1007/s10703-008-0048-7](https://doi.org/10.1007/s10703-008-0048-7). 48, 137
- Michael C. Browne, Edmund M. Clarke, and Orna Grumberg. Characterizing finite kripke structures in propositional temporal logic. *Theor. Comput. Sci.*, 59:115–131, 1988. DOI: [10.1016/0304-3975\(88\)90098-9](https://doi.org/10.1016/0304-3975(88)90098-9). 29
- Michael C. Browne, Edmund M. Clarke, and Orna Grumberg. Reasoning about networks with many identical finite state processes. *Information and Computation*, 81(1):13–31, 1989. DOI: [10.1016/0890-5401\(89\)90026-6](https://doi.org/10.1016/0890-5401(89)90026-6). 14
- Muffy Calder and Alice Miller. Using spin to analyse the tree identification phase of the ieee 1394 high-performance serial bus (firewire) protocol. *Formal Aspects of Computing*, 14(3):247–266, 2003. DOI: [10.1007/s001650300004](https://doi.org/10.1007/s001650300004). 131
- Roberto Cavada, Alessandro Cimatti, Michele Dorigatti, Alberto Griggio, Alessandro Mariotti, Andrea Micheli, Sergio Mover, Marco Roveri, and Stefano Tonetta. The nuXmv symbolic model checker. In *CAV*, volume 8559 of *LNCS*, pages 334–342, 2014. DOI: [10.1007/978-3-319-08867-9_22](https://doi.org/10.1007/978-3-319-08867-9_22). 139
- K. M. Chandy and J. Misra. The drinking philosophers problem. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 6(4):632–646, 1984. DOI: [10.1145/1780.1804.48](https://doi.org/10.1145/1780.1804.48)
- Allan Cheng, Javier Esparza, and Jens Palsberg. Complexity results for 1-safe nets. In *FSTTCS*, volume 761 of *LNCS*, pages 326–337. Springer, 1993. DOI: [10.1007/3-540-57529-4_66](https://doi.org/10.1007/3-540-57529-4_66). 105
- Alessandro Cimatti, Edmund M. Clarke, Enrico Giunchiglia, Fausto Giunchiglia, Marco Pistore, Marco Roveri, Roberto Sebastiani, and Armando Tacchella. NuSMV 2: An opensource tool for symbolic model checking. In *CAV*, volume 2404 of *LNCS*, pages 359–364, 2002. DOI: [10.1007/3-540-45657-0_29](https://doi.org/10.1007/3-540-45657-0_29). 130
- Edmund Clarke, Orna Grumberg, Hiromi Hiraishi, Somesh Jha, David Long, Kenneth McMillan, and Linda Ness. Verification of the futurebus+ cache coherence protocol. Technical report, DTIC Document, 1992. DOI: [10.1007/BF01383968](https://doi.org/10.1007/BF01383968). 1
- Edmund Clarke, Orna Grumberg, and Doron Peled. *Model Checking*. MIT Press, 1999. 1, 2, 13, 14, 129

148 BIBLIOGRAPHY

- Edmund Clarke, Muralidhar Talupur, Tayssir Touili, and Helmut Veith. Verification by network decomposition. In *CONCUR 2004*, volume 3170, pages 276–291, 2004. DOI: [10.1007/978-3-540-28644-8_18](https://doi.org/10.1007/978-3-540-28644-8_18). 2, 19, 25, 31, 40, 41, 42
- Edmund Clarke, Murali Talupur, and Helmut Veith. Proving ptolemy right: The environment abstraction framework for model checking concurrent systems. In *TACAS*, pages 33–47. Springer, 2008. DOI: [10.1007/978-3-540-78800-3_4](https://doi.org/10.1007/978-3-540-78800-3_4). 130
- Edmund M Clarke, Orna Grumberg, and Somesh Jha. Verifying parameterized networks using abstraction and regular languages. In *CONCUR*, pages 395–407. Springer, 1995. DOI: [10.1007/3-540-60218-6_30](https://doi.org/10.1007/3-540-60218-6_30). 136
- Edmund M. Clarke, Orna Grumberg, and Somesh Jha. Verifying parameterized networks. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 19(5):726–750, 1997. DOI: [10.1145/265943.265960](https://doi.org/10.1145/265943.265960). 136
- Sylvain Conchon, Amit Goel, Sava Krstic, Alain Mebsout, and Fatiha Zaïdi. Cubicle: A parallel smt-based model checker for parameterized systems - tool paper. In *CAV*, volume 7358 of *LNCS*, pages 718–724. Springer, 2012. DOI: [10.1007/978-3-642-31424-7_55](https://doi.org/10.1007/978-3-642-31424-7_55). 133, 139
- Giorgio Delzanno. Constraint-based verification of parameterized cache coherence protocols. *Formal Methods in System Design*, 23(3):257–301, 2003. DOI: [10.1023/A:1026276129010](https://doi.org/10.1023/A:1026276129010). 137
- Giorgio Delzanno and Riccardo Traverso. Decidability and complexity results for verification of asynchronous broadcast networks. In *LATA*, volume 7810 of *LNCS*, pages 238–249. Springer, 2013. DOI: [10.1007/978-3-642-37064-9_22](https://doi.org/10.1007/978-3-642-37064-9_22). 128
- Giorgio Delzanno, Jean-François Raskin, and Laurent Van Begin. Towards the automated verification of multithreaded Java programs. In *TACAS*, volume 2280 of *LNCS*, pages 173–187, 2002. DOI: [10.1007/3-540-46002-0_13](https://doi.org/10.1007/3-540-46002-0_13). 51
- Giorgio Delzanno, Arnaud Sangnier, and Gianluigi Zavattaro. Parameterized verification of ad hoc networks. In *CONCUR*, volume 6269 of *LNCS*, pages 313–327, 2010. DOI: [10.1007/978-3-642-15375-4_22](https://doi.org/10.1007/978-3-642-15375-4_22). 12, 19, 103, 107, 108, 109, 110, 111, 113, 114, 115, 120, 122, 124, 127
- Giorgio Delzanno, Arnaud Sangnier, and Gianluigi Zavattaro. On the power of cliques in the parameterized verification of ad hoc networks. In *FOSSACS*, volume 6604 of *LNCS*, pages 441–455. Springer, 2011. DOI: [10.1007/978-3-642-19805-2_30](https://doi.org/10.1007/978-3-642-19805-2_30). 19, 103, 107, 108, 109, 113, 114, 119, 120, 122, 127
- Giorgio Delzanno, Arnaud Sangnier, Riccardo Traverso, and Gianluigi Zavattaro. The cost of parameterized reachability in mobile ad hoc networks. *CoRR*, abs/1202.5850, 2012a. DOI: [10.1007/978-3-642-30793-5_15](https://doi.org/10.1007/978-3-642-30793-5_15). 126

- Giorgio Delzanno, Arnaud Sangnier, and Gianluigi Zavattaro. Verification of ad hoc networks with node and communication failures. In *FORTE*, volume 7273 of *LNCS*, pages 235–250. Springer, 2012b. DOI: [10.1007/978-3-642-41036-9_11](https://doi.org/10.1007/978-3-642-41036-9_11). 19, 103, 105, 106, 107, 108, 124, 128
- Giorgio Delzanno, Arnaud Sangnier, and Riccardo Traverso. Parameterized verification of broadcast networks of register automata. In *RP*, volume 8169 of *LNCS*, pages 109–121. Springer, 2013. DOI: [10.1007/3-540-61474-5_86](https://doi.org/10.1007/3-540-61474-5_86). 128
- David L Dill. The mur ϕ verification system. In *CAV*, pages 390–393. Springer, 1996. DOI: [10.1007/978-3-540-69850-0_5](https://doi.org/10.1007/978-3-540-69850-0_5). 130
- David L. Dill. A retrospective on murphi. In *25 Years of Model Checking - History, Achievements, Perspectives*, volume 5000 of *LNCS*, pages 77–88. Springer, 2008. DOI: [10.1002/jgt.3190140406](https://doi.org/10.1002/jgt.3190140406). 130
- Guoli Ding. Subgraphs and well-quasi-ordering. *Journal of Graph Theory*, 16(5):489–502, 1992. DOI: [10.1002/jgt.3190160509](https://doi.org/10.1002/jgt.3190160509). 119
- Jacob Elgaard, Nils Karlund, and Anders Møller. MONA 1.x: new techniques for WS1S and WS2S. In *CAV*, volume 1427 of *LNCS*, pages 516–520. Springer, 1998. DOI: [10.1007/BFb0028773](https://doi.org/10.1007/BFb0028773). 132, 139
- E. Allen Emerson and Edmund M. Clarke. Characterizing correctness properties of parallel programs using fixpoints. In *ICALP*, volume 85 of *LNCS*, pages 169–181. Springer, 1980. DOI: [10.1007/3-540-10003-2_69](https://doi.org/10.1007/3-540-10003-2_69). 1
- E. Allen Emerson and Vineet Kahlon. Reducing model checking of the many to the few. In *CADE*, volume 1831 of *LNCS*, pages 236–254. Springer Berlin Heidelberg, 2000. DOI: [10.1007/10721959_19](https://doi.org/10.1007/10721959_19). 19, 29, 65, 73, 74, 81, 82, 84, 86, 89, 95, 100, 101, 138
- E. Allen Emerson and Vineet Kahlon. Model checking large-scale and parameterized resource allocation systems. In *TACAS*, volume 2280 of *LNCS*, pages 251–265, 2002. DOI: [10.1007/3-540-46002-0_18](https://doi.org/10.1007/3-540-46002-0_18). 48
- E. Allen Emerson and Vineet Kahlon. Exact and efficient verification of parameterized cache coherence protocols. In *CHARME*, volume 2860 of *LNCS*, pages 247–262. Springer, 2003a. DOI: [10.1007/978-3-540-39724-3_22](https://doi.org/10.1007/978-3-540-39724-3_22). 19, 65, 74, 99, 101
- E. Allen Emerson and Vineet Kahlon. Model checking guarded protocols. In *LICS*, pages 361–370. IEEE, 2003b. DOI: [10.1109/LICS.2003.1210076](https://doi.org/10.1109/LICS.2003.1210076). 18, 19, 23, 51, 58, 59, 62, 73, 74, 76, 79, 81, 95, 96, 97, 98, 99, 100

150 BIBLIOGRAPHY

- E. Allen Emerson and Vineet Kahlon. Rapid parameterized model checking of snoopy cache coherence protocols. In *TACAS*, volume 2619 of *LNCS*, pages 144–159. Springer, 2003c. DOI: [10.1007/3-540-36577-X_11](https://doi.org/10.1007/3-540-36577-X_11). 19, 65, 74, 99, 101
- E. Allen Emerson and Vineet Kahlon. Parameterized model checking of ring-based message passing systems. In *CSL*, volume 3210 of *LNCS*, pages 325–339. Springer, 2004. DOI: [10.1007/978-3-540-30124-0_26](https://doi.org/10.1007/978-3-540-30124-0_26). 19, 44, 45, 48
- E. Allen Emerson and Kedar S. Namjoshi. Reasoning about rings. In *POPL*, pages 85–94, 1995. DOI: [10.1145/199448.199468](https://doi.org/10.1145/199448.199468). 2, 12, 14, 16, 19, 23, 24, 25, 29, 31, 35, 36, 38, 39, 40, 42, 84
- E. Allen Emerson and Kedar S. Namjoshi. Automatic verification of parameterized synchronous systems. In *CAV*, volume 1102 of *LNCS*, pages 87–98. Springer, 1996. DOI: [10.1007/3-540-61474-5_60](https://doi.org/10.1007/3-540-61474-5_60). 18, 73, 74, 77
- E. Allen Emerson and Kedar S. Namjoshi. On model checking for non-deterministic infinite-state systems. In *LICS*, pages 70–80. IEEE Computer Society, 1998. DOI: [10.1109/LICS.1998.705644](https://doi.org/10.1109/LICS.1998.705644). 18, 19, 100
- E. Allen Emerson and Kedar S. Namjoshi. On reasoning about rings. *Int. J. Found. Comput. Sci.*, 14(4):527–550, 2003. DOI: [10.1142/S0129054103001881](https://doi.org/10.1142/S0129054103001881). 12, 14, 16, 19, 24, 25, 29, 31, 35, 36, 38, 39, 40, 42, 43
- E. Allen Emerson and A. Prasad Sistla. Symmetry and model checking. *Formal Methods in System Design*, 9:105–131, 1996. DOI: [10.1007/BF00625970](https://doi.org/10.1007/BF00625970). 25
- Javier Esparza. Decidability and complexity of petri net problems - an introduction. In *In Lectures on Petri Nets I: Basic Models*, pages 374–428. Springer-Verlag, 1998. DOI: [10.1007/3-540-65306-6_20](https://doi.org/10.1007/3-540-65306-6_20). 25, 27
- Javier Esparza. Keeping a crowd safe: On the complexity of parameterized verification. In *STACS*, 2014. DOI: [10.4230/LIPIcs.STACS.2014.1](https://doi.org/10.4230/LIPIcs.STACS.2014.1). 63, 64, 143
- Javier Esparza and Mogens Nielsen. Decidability issues for petri nets - a survey. *Bulletin of the EATCS*, 52:244–262, 1994. 95
- Javier Esparza, Alain Finkel, and Richard Mayr. On the verification of broadcast protocols. *LICS*, page 352, 1999. ISSN 1043-6871. DOI: [10.1109/LICS.1999.782630](https://doi.org/10.1109/LICS.1999.782630). 10, 19, 51, 58, 59, 99, 100, 108, 114
- Javier Esparza, Pierre Ganty, and Rupak Majumdar. Parameterized verification of asynchronous shared-memory systems. In *CAV*, pages 124–140, 2013. DOI: [10.1007/978-3-642-39799-8_8](https://doi.org/10.1007/978-3-642-39799-8_8). 137

- Johannes Faber, Carsten Ihleemann, Swen Jacobs, and Viorica Sofronie-Stokkermans. Automatic verification of parametric specifications with complex topologies. In *IFM*, volume 6396 of *LNCS*, pages 152–167. Springer, 2010. DOI: [10.1007/978-3-642-16265-7_12](https://doi.org/10.1007/978-3-642-16265-7_12). 134
- Yi Fang, Kenneth L. McMillan, Amir Pnueli, and Lenore D. Zuck. Liveness by invisible invariants. In *FORTE*, pages 356–371, 2006. DOI: [10.1007/11888116_26](https://doi.org/10.1007/11888116_26). 137
- Azadeh Farzan, Zachary Kincaid, and Andreas Podelski. Proofs that count. In *POPL*, pages 151–164, 2014. DOI: [10.1145/2535838.2535885](https://doi.org/10.1145/2535838.2535885). 138
- Azadeh Farzan, Matthias Heizmann, Jochen Hoenicke, Zachary Kincaid, and Andreas Podelski. Automated program verification. In *LATA*, pages 25–46, 2015a. DOI: [10.1007/978-3-319-15579-1_2](https://doi.org/10.1007/978-3-319-15579-1_2). 138
- Azadeh Farzan, Zachary Kincaid, and Andreas Podelski. Proof spaces for unbounded parallelism. In *POPL*, pages 407–420. ACM, 2015b. DOI: [10.1145/2676726.2677012](https://doi.org/10.1145/2676726.2677012). 138
- Alain Finkel and Philippe Schnoebelen. Well-structured transition systems everywhere! *Theoretical Computer Science*, 256(1–2):63–92, 2001. DOI: [10.1016/S0304-3975\(00\)00102-X](https://doi.org/10.1016/S0304-3975(00)00102-X). 25, 26, 122
- Matthias Függer and Josef Widder. Efficient checking of link-reversal-based concurrent systems. In *CONCUR*, volume 7454 of *LNCS*, pages 486–499, 2012. DOI: [10.1007/978-3-642-32940-1_34](https://doi.org/10.1007/978-3-642-32940-1_34). 48
- Pierre Ganty and Ahmed Rezine. Ordered counter-abstraction — refinable subword relations for parameterized verification. In *LATA*, volume 8370 of *LNCS*, pages 396–408. Springer, 2014. DOI: [10.1007/978-3-319-04921-2_32](https://doi.org/10.1007/978-3-319-04921-2_32). 133, 139
- Steven M. German and A. Prasad Sistla. Reasoning about systems with many processes. *J. ACM*, 39(3):675–735, 1992. ISSN 0004-5411. DOI: [10.1145/146637.146681](https://doi.org/10.1145/146637.146681). 12, 18, 19, 29, 51, 52, 54, 55, 58, 59, 64, 99, 143
- Silvio Ghilardi and Silvio Ranise. Backward reachability of array-based systems by SMT solving: Termination and invariant synthesis. *Logical Methods in Computer Science*, 6(4), 2010. DOI: [10.2168/LMCS-6\(4:10\)2010](https://doi.org/10.2168/LMCS-6(4:10)2010). 133, 139
- Silvio Ghilardi, Enrica Nicolini, Silvio Ranise, and Daniele Zucchelli. Towards SMT model checking of array-based systems. In *Automated Reasoning*, volume 5195 of *LNCS*, pages 67–82. Springer, 2008. DOI: [10.1007/978-3-540-71070-7_6](https://doi.org/10.1007/978-3-540-71070-7_6). 133
- Jim Gray. Notes on data base operating systems. In *Operating Systems, An Advanced Course*, volume 60 of *LNCS*, pages 393–481. Springer, 1978. DOI: [10.1007/3-540-08755-9_9](https://doi.org/10.1007/3-540-08755-9_9). 103

152 BIBLIOGRAPHY

- Orna Grumberg and Helmut Veith, editors. *25 Years of Model Checking – History, Achievements, Perspectives*, volume 5000 of *LNCS*, 2008. [1](#)
- Jesper G. Henriksen, Jakob L. Jensen, Michael E. Jørgensen, Nils Klarlund, Robert Paige, Theis Rauhe, and Anders Sandholm. Mona: Monadic second-order logic in practice. In *TACAS*, volume 1019 of *LNCS*, pages 89–110. Springer, 1995. DOI: [10.1007/3-540-60630-0_5](https://doi.org/10.1007/3-540-60630-0_5). [132](#)
- Graham Higman. Ordering by divisibility in abstract algebras. *Proceedings of the London Mathematical Society*, s3-2(1):326–336, 1952. DOI: [10.1112/plms/s3-2.1.326](https://doi.org/10.1112/plms/s3-2.1.326). [119](#)
- Gerard Holzmann. *The SPIN Model Checker: Primer and Reference Manual*. Addison-Wesley Professional, 2003. [130](#)
- C. Norris Ip and David L. Dill. Verifying systems with replicated components in Murφ. *Formal Methods in System Design*, 14(3):273–310, 1999. DOI: [10.1023/A:1008723125149](https://doi.org/10.1023/A:1008723125149). [130](#)
- Swen Jacobs and Viorica Sofronie-Stokkermans. Applications of hierarchical reasoning in the verification of complex systems. *Electr. Notes Theor. Comput. Sci.*, 174(8):39–54, 2007. DOI: [10.1016/j.entcs.2006.11.038](https://doi.org/10.1016/j.entcs.2006.11.038). [134](#)
- Henrik Jensen and Nancy Lynch. A proof of Burns n-process mutual exclusion algorithm using abstraction. In *TACAS*, volume 1384 of *LNCS*, pages 409–423. Springer, 1998. DOI: [10.1007/BFb0054186](https://doi.org/10.1007/BFb0054186). [131](#)
- Annu John, Igor Konnov, Ulrich Schmid, Helmut Veith, and Josef Widder. Parameterized model checking of fault-tolerant distributed algorithms by abstraction. In *FMCAD*, pages 201–209, 2013. DOI: [10.1145/2484239.2484285](https://doi.org/10.1145/2484239.2484285). [3](#), [129](#), [130](#), [139](#)
- Taylor T. Johnson. *Uniform Verification of Safety for Parameterized Networks of Hybrid Automata*. PhD thesis, University of Illinois at Urbana-Champaign, Urbana, IL 61801, 2013. [137](#), [139](#)
- Taylor T. Johnson and Sayan Mitra. A small model theorem for rectangular hybrid automata networks. In *FORTE*, pages 18–34, 2012. DOI: [10.1007/978-3-642-30793-5_2](https://doi.org/10.1007/978-3-642-30793-5_2). [137](#)
- Alexander Kaiser, Daniel Kroening, and Thomas Wahl. Dynamic cutoff detection in parameterized concurrent programs. In *CAV*, volume 6174 of *LNCS*, pages 654–659. Springer, 2010. DOI: [10.1007/978-3-642-14295-6_55](https://doi.org/10.1007/978-3-642-14295-6_55). [134](#)
- Holger Karl. *Protocols and architectures for wireless sensor networks*. Wiley, Hoboken, NJ, 2005. ISBN 978-0-470-09510-2. DOI: [10.1002/0470095121](https://doi.org/10.1002/0470095121). [103](#), [107](#)
- Yonit Kesten and Amir Pnueli. Control and data abstraction: The cornerstones of practical formal verification. *International Journal on Software Tools for Technology Transfer*, 2(4):328–342, 2000. DOI: [10.1007/s100090050040](https://doi.org/10.1007/s100090050040). [136](#)

- Yonit Kesten, Oded Maler, Monica Marcus, Amir Pnueli, and Elad Shahar. Symbolic model checking with rich assertional languages. In *CAV*, pages 424–435. Springer, 1997. DOI: [10.1007/3-540-63166-6_41](https://doi.org/10.1007/3-540-63166-6_41). 1, 132
- Ayrat Khalimov, Swen Jacobs, and Roderick Bloem. Towards efficient parameterized synthesis. In *VMCAI*, volume 7737 of *LNCS*, pages 108–127. Springer, 2013a. DOI: [10.1007/978-3-642-35873-9_9](https://doi.org/10.1007/978-3-642-35873-9_9). 38, 40
- Ayrat Khalimov, Swen Jacobs, and Roderick Bloem. PARTY parameterized synthesis of token rings. In *CAV*, volume 8044 of *LNCS*, pages 928–933. Springer, 2013b. DOI: [10.1007/978-3-642-39799-8_66](https://doi.org/10.1007/978-3-642-39799-8_66). 139
- Igor V. Konnov and Vladimir A. Zakharov. An invariant-based approach to the verification of asynchronous parameterized networks. *J. Symb. Comput.*, 45(11):1144–1162, 2010. DOI: [10.1016/j.jsc.2008.11.006](https://doi.org/10.1016/j.jsc.2008.11.006). 136, 139
- Panagiotis Kouvaros and Alessio Lomuscio. Automatic verification of parameterised multi-agent systems. In *AAMAS*, pages 861–868, 2013a. 64
- Panagiotis Kouvaros and Alessio Lomuscio. A cutoff technique for the verification of parameterised interpreted systems with parameterised environments. In *IJCAI*, 2013b. 64
- Panagiotis Kouvaros and Alessio Lomuscio. A counter abstraction technique for the verification of robot swarms. In *AAAI Conference on Artificial Intelligence*, pages 2081–2088, 2015. 64
- Robert P Kurshan and Ken McMillan. A structural induction theorem for processes. In *Proceedings of the eighth annual ACM Symposium on Principles of distributed computing*, pages 239–247. ACM, 1989. DOI: [10.1145/72981.72998](https://doi.org/10.1145/72981.72998). 135, 136
- Leslie Lamport. Checking a multithreaded algorithm with +CAL. In *Distributed Computing*, pages 151–163. Springer, 2006. DOI: [10.1007/11864219_11](https://doi.org/10.1007/11864219_11). 2
- Axel Legay. T(O)RMC: A tool for (omega)-regular model checking. In *CAV*, volume 5123 of *LNCS*, pages 548–551. Springer, 2008. DOI: [10.1007/978-3-540-70545-1_52](https://doi.org/10.1007/978-3-540-70545-1_52). 132, 139
- Jérôme Leroux and Grégoire Sutre. Flat counter automata almost everywhere! In *ATVA*, volume 3707 of *LNCS*, pages 489–503, 2005. DOI: [10.1007/11562948_36](https://doi.org/10.1007/11562948_36). 137
- David Lesens, Nicolas Halbwachs, and Pascal Raymond. Automatic verification of parameterized linear networks of processes. In *POPL*, pages 346–357. ACM, 1997. DOI: [10.1145/263699.263747](https://doi.org/10.1145/263699.263747). 136
- Nancy Lynch. *Distributed Algorithms*. Morgan Kaufman Publishers, Inc., San Francisco, USA, 1996. 14, 144

154 BIBLIOGRAPHY

- Monika Maidl. A unifying model checking approach for safety properties of parameterized systems. In *CAV*, pages 311–323. Springer, 2001. DOI: [10.1007/3-540-44585-4_29](https://doi.org/10.1007/3-540-44585-4_29). 132
- Kenneth L. McMillan and Lenore D. Zuck. Invisible invariants and abstract interpretation. In *SAS*, pages 249–262, 2011. DOI: [10.1007/978-3-642-23702-7_20](https://doi.org/10.1007/978-3-642-23702-7_20). 137
- Robin Milner. *Communication and concurrency*. PHI Series in computer science. Prentice Hall, 1989. ISBN 978-0-13-115007-2. 3, 31
- Marvin L. Minsky. *Computation: finite and infinite machines*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1967. ISBN 0-13-165563-9. 23, 24
- André Platzer. A complete axiomatization of quantified differential dynamic logic for distributed hybrid systems. *Logical Methods in Computer Science*, 8(4), 2012. DOI: [10.2168/LMCS-8\(4:17\)2012](https://doi.org/10.2168/LMCS-8(4:17)2012). 139
- André Platzer and Jan-David Quesel. Keymaera: A hybrid theorem prover for hybrid systems (system description). In *IJCAR*, volume 5195 of *LNCS*, pages 171–178. Springer, 2008. DOI: [10.1007/978-3-540-71070-7_15](https://doi.org/10.1007/978-3-540-71070-7_15). 139
- Amir Pnueli and Elad Shahar. A platform for combining deductive with algorithmic verification. In *CAV*, volume 1102 of *LNCS*, pages 184–195. Springer, 1996. DOI: [10.1007/3-540-61474-5_68](https://doi.org/10.1007/3-540-61474-5_68). 132, 139
- Amir Pnueli and Elad Shahar. Liveness and acceleration in parameterized verification. In *CAV*, pages 328–343, 2000. DOI: [10.1007/10722167_26](https://doi.org/10.1007/10722167_26). 132
- Amir Pnueli, Sitvanit Ruah, and Lenore Zuck. Automatic deductive verification with invisible invariants. In *TACAS*, volume 2031 of *LNCS*, pages 82–97. Springer, 2001. DOI: [10.1007/3-540-45319-9_7](https://doi.org/10.1007/3-540-45319-9_7). 137
- Amir Pnueli, Jessie Xu, and Lenore Zuck. Liveness with $(0,1,\infty)$ - counter abstraction. In *CAV*, volume 2404 of *LNCS*, pages 93–111. Springer, 2002. DOI: [10.1007/3-540-45657-0_9](https://doi.org/10.1007/3-540-45657-0_9). 3, 129, 130
- Amir Pnueli, Yaniv Sa’ar, and Lenore D. Zuck. JTLV: A framework for developing verification algorithms. In *CAV*, volume 6174 of *LNCS*, pages 171–174. Springer, 2010. 132, 139
- Jean-Pierre Queille and Joseph Sifakis. Specification and verification of concurrent systems in cesar. In *International Symposium on Programming*, volume 137 of *LNCS*, pages 337–351. Springer, 1982. DOI: [10.1007/3-540-11494-7_22](https://doi.org/10.1007/3-540-11494-7_22). 1
- Tatiana Rybina and Andrei Voronkov. BRAIN : Backward reachability analysis with integers. In *AMAST*, volume 2422 of *LNCS*, pages 489–494. Springer, 2002. DOI: [10.1007/978-3-642-14295-6_18](https://doi.org/10.1007/978-3-642-14295-6_18). 139

BIBLIOGRAPHY 155

- Sylvain Schmitz and Philippe Schnoebelen. The power of well-structured systems. In *CONCUR*, volume 8052 of *LNCS*, pages 5–24. Springer, 2013. Invited paper. DOI: [10.1007/978-3-642-40184-8_2](https://doi.org/10.1007/978-3-642-40184-8_2). 63, 143
- Ze'ev Shtadler and Orna Grumberg. Network grammars, communication behaviors and automatic verification. In *Automatic Verification Methods for Finite State Systems*, pages 151–165. Springer, 1990. DOI: [10.1007/3-540-52148-8_13](https://doi.org/10.1007/3-540-52148-8_13). 136
- T.K. Srikanth and Sam Toueg. Simulating authenticated broadcasts to derive simple fault-tolerant algorithms. *Distributed Computing*, 2:80–94, 1987. DOI: [10.1007/BF01667080](https://doi.org/10.1007/BF01667080). 3, 14
- Ichiro Suzuki. Proving properties of a ring of finite-state machines. *Inf. Process. Lett.*, 28(4): 213–214, July 1988. DOI: [10.1016/0020-0190\(88\)90211-6](https://doi.org/10.1016/0020-0190(88)90211-6). 3, 23, 31, 42
- Murali Talupur and Mark R. Tuttle. Going with the flow: Parameterized verification using message flows. In *FMCAD*, pages 1–8. IEEE, 2008. DOI: [10.1109/FMCAD.2008.ECP.14](https://doi.org/10.1109/FMCAD.2008.ECP.14). 131
- Anthony Widjaja To and Leonid Libkin. Algorithmic metatheorems for decidable LTL model checking over infinite systems. In *Foundations of Software Science and Computational Structures*, pages 221–236. Springer, 2010. DOI: [10.1007/978-3-642-12032-9_16](https://doi.org/10.1007/978-3-642-12032-9_16). 132
- Moshe Y. Vardi. An automata-theoretic approach to linear temporal logic. In *Banff Higher Order Workshop*, pages 238–266, 1995. DOI: [10.1007/3-540-60915-6_6](https://doi.org/10.1007/3-540-60915-6_6). 55
- Pierre Wolper and Vinciane Lovinfosse. Verifying properties of large sets of processes with network invariants. In *Automatic Verification Methods for Finite State Systems*, volume 407 of *LNCS*, pages 68–80, 1989. DOI: [10.1007/3-540-52148-8_6](https://doi.org/10.1007/3-540-52148-8_6). 3, 135, 136
- Tuba Yavuz-Kahveci and Tevfik Bultan. Action language verifier: an infinite-state model checker for reactive software specifications. *Formal Methods in System Design*, 35(3):325–367, 2009. DOI: [10.1007/s10703-009-0081-1](https://doi.org/10.1007/s10703-009-0081-1). 139
- Hsu-Chun Yen. A unified approach for deciding the existence of certain petri net paths. *Information and Computation*, 96(1):119 – 137, 1992. DOI: [10.1016/0890-5401\(92\)90059-O](https://doi.org/10.1016/0890-5401(92)90059-O). 25

Authors' Biographies

RODERICK BLOEM

Roderick Bloem is a professor at Graz University of Technology. He received an M.Sc. in computer science from Leiden University in the Netherlands (1996) and a Ph.D. from the University of Colorado at Boulder (2001). His thesis work, under the supervision of Fabio Somenzi, was on formal verification using Linear Temporal Logic.

Since 2002, he has been an assistant professor at Graz University of Technology and a full professor since 2008. His research interests are in formal methods for the design and verification of digital systems, including hardware, software, and combinations such as embedded systems. He studies applications of game theory to the automatic synthesis of systems from their specifications, connections between temporal logics and omega-automata, model checking, and automatic fault localization and repair.

SWEN JACOBS

Swen Jacobs is a postdoc at Saarland University. He received his Ph.D. (Dr. Ing.) from Saarland University for his work on decision procedures for the verification of complex systems at the Max-Planck-Institute for Informatics.

He worked at École Polytechnique Fédérale de Lausanne (EPFL), at Technical University Graz, and has been a visiting professor at the University of Ljubljana. His current work focuses on the automated verification and synthesis of distributed systems, based on a combination of logical and game-theoretic methods.

AYRAT KHALIMOV

Ayrat Khalimov is a Ph.D. student at Technical University of Graz, Austria. He received his master's degree in applied physics and Mathematics at Moscow Institute of Physics and Technology (MIPT), with the thesis focusing on a method of calculation of current leakages in hardware circuits. Later, he joined Dependable Systems Lab at École Polytechnique Fédérale de Lausanne (EPFL) for an internship where he researched symbolic execution techniques for software verification. His current area of research is parameterized synthesis and verification.

IGOR KONNOV

Igor Konnov is a postdoc (Universitätsassistent) at the Formal Methods in Systems Engineering Group, Institute of Information Systems of TU Wien (Vienna University of Technology). His research interests include model checking, parameterized model checking, and verification of distributed algorithms.

He received his Specialist (comparable to M.Sc.) and Ph.D. degrees in applied mathematics and computer science from Lomonosov Moscow State University. In his Ph.D. thesis, he introduced new techniques for parameterized model checking.

SASHA RUBIN

Sasha Rubin is a postdoc at the Università degli Studi di Napoli “Federico II” (University of Naples). He is broadly interested in the connections between automata theory and logic, and in particular, in formal verification, game theory, and finite model theory. He received his Ph.D. from the University of Auckland and was awarded the Vice-chancellor’s prize for the best doctoral thesis in the Faculty of Science. He previously held a New Zealand Science and Technology Postdoctoral Fellowship. He is currently a Marie Curie fellow of the Istituto Nazionale di Alta Matematica.

HELMUT VEITH

Helmut Veith is a professor at the Faculty of Informatics of TU Wien (Vienna University of Technology), and an adjunct professor at Carnegie Mellon University. He has a diploma in Computational Logic and a Ph.D. sub auspiciis praesidentis in Computer Science, both from TU Wien. Prior to his appointment to Vienna, he held professor positions at TU Darmstadt and TU Munich.

In his research, Helmut Veith applies formal and logical methods to problems in software technology and engineering. His current work focuses on model checking, software verification and testing, embedded software, and computer security.

JOSEF WIDDER

Josef Widder is an assistant professor (Privatdozent) at the Faculty of Informatics of TU Wien. His primary area of interest is the theoretical approach to distributed algorithms, currently focusing on automated verification of fault-tolerant distributed algorithms.

He received his Ph.D. (Doctor technicae), and his habilitation in computer science from TU Wien. He worked at the Embedded Computing Systems group and the Formal Methods in Systems Engineering group of TU Wien (Austria), the Laboratoire d’Informatique de l’Ecole polytechnique (France), and the Parasol Lab at Texas A&M University (USA).

Multi-Agent Path Planning in Known Dynamic Environments^{*}

Aniello Murano¹, Giuseppe Perelli², Sasha Rubin¹

¹Università di Napoli “Federico II” and ²University of Oxford

Abstract. We consider the problem of planning paths of multiple agents in a dynamic but predictable environment. Typical scenarios are evacuation, reconfiguration, and containment. We present a novel representation of abstract path-planning problems in which the stationary environment is explicitly coded as a graph (called the arena) while the dynamic environment is treated as just another agent. The complexity of planning using this representation is PSPACE-complete. The arena complexity (i.e., the complexity of the planning problem in which the graph is the only input, in particular, the number of agents is fixed) is NP-hard. Thus, we provide structural restrictions that put the arena complexity of the planning problem into PTIME (for any fixed number of agents). The importance of our work is that these structural conditions (and hence the complexity results) do not depend on graph-theoretic properties of the arena (such as clique- or tree-width), but rather on the abilities of the agents.

1 Introduction

The path-planning problem is to find collision-free paths for mobile agents in an environment that may contain obstacles [20, 32, 10, 27]. Obstacles, which may not always be stationary, are *known* if their size, locations, motions, etc. are fixed ahead of planning.

For example, consider a building consisting of rooms, corridors between certain pairs of rooms, and a set of exits. Initially, a fixed number of people are positioned in various rooms, and a flood begins in one of the rooms. At each time step every agent can either stay where it is or move through a corridor to an adjacent room. Suppose the flood spreads radially (i.e., at each time step it reaches all adjacent rooms that are accessible via a corridor). The path planning problem is to exhibit a sequence of actions for the agents that ensures they can reach an exit before the flood traps them.

Other applications are to space exploration, warehouse management, intelligent transportation, assembly or disassembly, and computer games (see [16] and the references therein).

* This work has been partially supported by the FP7 EU project 600958-SHERPA and the ERC Advanced Grant “RACE” (291528) at Oxford. Sasha Rubin is a Marie Curie fellow of the Istituto Nazionale di Alta Matematica.

The AI literature on planning has established that planning is intractable, e.g., propositional STRIPS planning is PSPACE-COMPLETE [2]. To gain deeper insight into what makes planning hard, one must study structural properties of the problem, e.g., [19]:

For many discrete planning problems that we would like a computer to solve, the state space is enormous (e.g., 10^{100} states). Therefore, substantial effort has been invested in constructing implicit encodings of problems in hopes that the entire state space does not have to be explored by the algorithm to solve the problem.

In this paper we propose an implicit representation (“encoding” in the language above) of path-planning problems of mobile agents in which obstacles are known, and agents should collaborate, rather than compete, to achieve some goal. Known but dynamic environments, such as the flood in the example above, are treated as agents.

Since our representation is exponentially compact, the associated decision problem is PSPACE-complete (Theorems 1 and 2). This is true even for a fixed number of agents, i.e., this high space complexity is not the result of the availability of more and more agents. The *restricted path-planning problem* assumes that all the data is fixed in advance, except for the arena which is the input. We prove that the complexity of the restricted path-planning problem, called the *arena complexity* in the abstract, may be NP-hard (Proposition 1); it is not known if it can be PSPACE-hard. Thus, we describe cases that are solvable in PTIME in which the agent behaviour is restricted (Theorem 3): every agent is either monotone or of bounded-extent. Informally, an agent is monotone if its set of positions can only expand over time (or, only shrink over time). A typical example is the flood: once a room is flooded it stays flooded. An agent is of bounded-extent if it can only occupy a set of positions of bounded size (where the bound does not depend on the size of the arena). A typical example are people: each person in the building can occupy only one room at a time, no matter how large the room is.

1.1 Related Work

Much work in robotics focuses on geometric reasoning, e.g., [32, 30, 18]. Although our environment is discrete (i.e., a finite graph), it may represent a discretisation of the geometric structure of the environment, e.g., a vertex may represent an area not occupied by any of the obstacles. For a discussion of the subtle issues involved in such a translation, see for instance [16].

Standard ways to deal with the fact that planning has, in general, high computational complexity, is to use abstractions and heuristics, see for instance a recent survey on path planning [27]. In contrast, in this paper we isolate computationally tractable but interesting path-planning scenarios.

Standard ways to encode planning problems are logic-based: e.g., the situation calculus is a first-order logic in which the main objects are finite sequences of

actions; and in the set-theoretical representation (such as STRIPS or the multi-agent extension MA-STRIPS [1]) every state is an evaluation of a fixed number of Boolean variables [10]. In contrast, our representation is graph-theoretic and represents the positions of the agents on a graph. Although our planning problems can be expressed in these logical formalisms, this would hide the graphical nature of the problem, see Section 5.

Our representation is related to multi-robot path-planning problems and puzzles [15, 23, 8, 28, 26]. In [15, 28, 26] the goal is to rearrange a group of robots in a given graph, without colliding with each other or with obstacles. The non-optimal version of that problem is in PTIME [15]. We can encode the variation in which more than one agent may move at a time [28], see Example 2. The motion-planning problems and puzzles of [12, 8] are PSPACE-complete. We use one such puzzle to prove a PSPACE-hard lower bound on our path-planning problems, see Theorem 2.

Monotonicity has been used to get PTIME algorithms in the setting of propositional temporal planning [4]. That work studies a propositional representation of planning problems in which the fluents (i.e., literals) are required to be monotone, e.g., once removed a fluent can never be added in any plan of the planning problem. In contrast, the natural translation of our representation into the propositional planning encoding does not preserve monotonicity, see Section 4.

Most of the planning literature, including this paper, focuses on attainment goals of the form “a target configuration can be reached from some initial configuration” [19]. In particular then, we can also model goals of the form “every agent eventually reaches its target vertex in a collision-free way”.

Our formalisation and results are inspired by formal methods. Other work in planning with the similar inspirations are an automata-theoretic approach to planning [5], and planning as model checking [24, 11, 31, 25, 21, 13]. These papers also supply centralised planning algorithms, however their representation is based on transition-systems, and hence is not compact, as ours is.

2 Representation of the Path-Planning Problem

We describe how we represent the path-planning problem. Informally, all moving entities (people, floods, fires) are considered to be agents. Agents operate in a finite graph (V, E) . At any given time, an agent can occupy a subset V , called its position (e.g., a person occupies a single vertex, and a flood may occupy more than one vertex). One may specify the legal positions \mathcal{L} of the agents, e.g., that agents cannot occupy overlapping vertices. An agent’s movements are governed by its mobility relation Δ that relates its current position to its next possible position (the fact that Δ is a relation, rather than a function, models that moves may be nondeterministic). In Section 3 we will discuss how to specify \mathcal{L} and Δ (in particular these objects can be defined algorithmically, or by formulas, e.g., of first-order logic).

Formally, let $k \in \mathbb{N}$ (representing the number the agents). An *arena* is a finite directed graph $\mathcal{A} = (V, E)$. Subsets of V are called *positions*. A $(k\text{-})$ *configuration*

(over \mathcal{A}) is an expansion of \mathcal{A} by k many positions, i.e., $\langle V, E, P_1, \dots, P_k \rangle$ where each $P_i \subseteq V$ is called the *position of agent i (in the configuration)*. Note that an agent can be in more than one vertex, e.g., a flood. A *mobility relation* (over \mathcal{A}) is a subset Δ of $2^V \times 2^V$ such that $(X, Y) \in \Delta$ implies $Y \subseteq X \cup E(X)$, where $E(X) := \{v \in V \mid \exists u \in X. (u, v) \in E\}$. The idea is that $(X, Y) \in \Delta$ means that a player can move from position X to position Y but only along edges.

A (*path-*)*planning domain* is a tuple $\mathcal{D} = \langle \mathcal{A}, \mathcal{L}, \Delta_1, \dots, \Delta_k \rangle$ where $\mathcal{A} = (V, E)$ is an arena, \mathcal{L} is a set of k -configurations of \mathcal{A} , called the *legal configurations*, and each Δ_i is a mobility relation over \mathcal{A} . For k -configurations c, d over \mathcal{A} write $c \vdash d$ to mean that d results from c via simultaneous application of Δ_i s, formally: for $c = \langle V, E, P_1, \dots, P_k \rangle$ and $d = \langle V, E, Q_1, \dots, Q_k \rangle$ define $c \vdash d$ iff $(P_i, Q_i) \in \Delta_i$ for all $i \leq k$. An *execution starting in configuration c* is a finite or infinite sequence π of configurations of \mathcal{A} such that $\pi_1 = c$ and for all i , a) $\pi_i \in \mathcal{L}$, and b) $\pi_i \vdash \pi_{i+1}$.

A (*path*) *planning instance* is a tuple $\mathcal{P} = \langle \mathcal{D}, \mathcal{I}, \mathcal{G} \rangle$ where \mathcal{I} and \mathcal{G} are sets of k -configurations over \mathcal{A} called the *initial configurations* and *goal configurations*. An execution π of \mathcal{P} is called a *solution* if $\pi_1 \in \mathcal{I}$ and there exists j such that $\pi_j \in \mathcal{G}$.

Example 1 (Evacuation). To model the flood example from the introduction with one person take $k = 2$, i.e., one agent is a person, and the other agent is the flood. Consider an initial configuration of the form $\langle V, E, \{p\}, \{f\} \rangle$, i.e., the person starts in some vertex s , and the source of the flood is in some vertex f . The goal configurations are of the form $\cup_{X \subseteq V} \langle V, E, \{t\}, X \rangle$ for some vertex t . The mobility of the person relates $\{v\}$ to all those $\{w\}$ such that $(v, w) \in E$ or $v = w$, i.e., the person can move to an adjacent position, or stay where it is. The mobility of the flood relates X to the single set $X \cup E(X)$, i.e., the flood expands radially. Thus a typical configuration in an execution is of the form $\langle V, E, \{v\}, F \rangle$ for some $v \in V$ and $F \subseteq V$. Finally, to specify that the person cannot move into a flooded area, define the set of legal configurations \mathcal{L} to be those of the form $\langle V, E, \{v\}, F \rangle$ such that $v \notin F$.

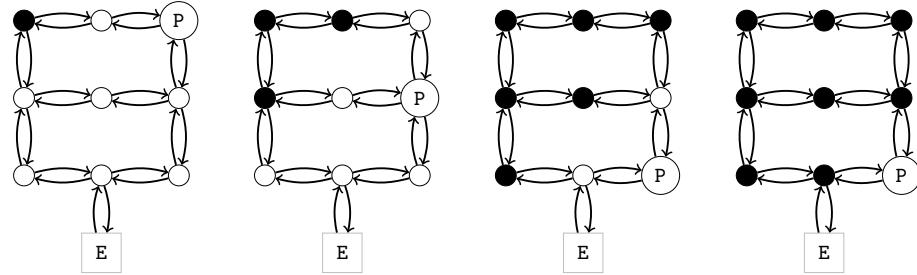


Fig. 1. An evolution of a flood scenario. Player P cannot reach the exit E

In Figure 1 we represent a possible evolution of an instance of the Evacuation scenario. The graph represents three floors of a building, connected by two lifts

located at the sides. Each floor has three connected rooms. Moreover, there is an exit point E at the basement, that is reachable from the central room of the first floor. In the initial configuration, the flood is located in the leftmost top room of the building, while Player P is in the rightmost top. At each step in time, the flood spreads from flooded rooms to the connected unflooded rooms, while P moves to some available room. Unfortunately for P , he gets stuck after three units of time without reaching the exit point. It is not hard to see that, in case the initial position of P is on a side of the second floor (or anywhere in the first floor), he can safely reach the exit point.

Example 2 (Reconfiguration). The abstract path-planning problem requires a set of k mobile robots to re-arrange themselves into a target configuration without colliding with each other or fixed obstacles [28]. This can be modeled as follows. There is one initial configuration, say $\langle V, E, \{v_1\}, \dots, \{v_k\} \rangle$, and one goal configuration, say $\langle V, E, \{t_1\}, \dots, \{t_k\} \rangle$. The mobility relation for each agent is the same as the mobility relation for the person in the previous example. Define the set of legal configurations \mathcal{L} to be those of the form $\langle V, E, \{v_1\}, \dots, \{v_k\} \rangle$ such that $v_i \neq v_j$ for $i \neq j$. This captures the fact that players should not collide.

Example 3 (Containment). Consider a variation of the flooding example in which the goal of the people is to erect barriers in some of the rooms in order to stop the flood from reaching a certain vertex (or cover a set of vertices). This can be modeled by having extra agents that represent barriers. Each barrier is associate with exactly one agent. When a barrier is placed on an unflooded vertex then the flood cannot enter that vertex (this can be coded in the legal-configurations). Agents can carry barriers with them or drop them when they move rooms (this is also expressed in the legal-configurations).

3 Complexity of the Path-Planning Problem

In order to talk about the decision problem for path-planning, we need a way to specify how the environment evolves no matter the size of the arena. For instance, a flood expands radially in any arena. We formalize this as follows.

A *mobility operator* is a function F that maps an arena \mathcal{A} to a mobility relation $F(\mathcal{A})$ over \mathcal{A} . For $k \in \mathbb{N}$, a k -*configuration operator* is a function C that maps an arena \mathcal{A} to a set $C(\mathcal{A})$ of k -configurations over \mathcal{A} . Informally, an operator is *tractable* if there is an efficient algorithm that computes the relation it induces. Formally, a mobility operator F is called *tractable* if there is a polynomial time algorithm that given an arena $\mathcal{A} = \langle V, E \rangle$, and a pair $(X, Y) \in 2^V \times 2^V$, decides whether $(X, Y) \in F(\mathcal{A})$. Similarly, a configuration operator is *tractable* if there is a polynomial time algorithm that, given an arena $\mathcal{A} = \langle V, E \rangle$ and a configuration $c = \langle V, E, P_1, \dots, P_k \rangle$ over \mathcal{A} decides whether $c \in C(\mathcal{A})$. Note that the operators in Examples 1 and 2 are tractable.

We assume, for the rest of this paper, that all operators are tractable.

Remark 1. This tractability assumption implies \vdash is tractable, i.e., there is a PTIME algorithm that given \mathcal{A} and k -configurations c, d over \mathcal{A} , decides whether $c \vdash d$.

Observe that fixing the following *planning data*

- $k \in \mathbb{N}$,
- mobility operators F_i (for each $i \leq k$),
- configuration operators L, I, G , and
- an arena $\mathcal{A} = \langle V, E \rangle$,

uniquely determines a path-planning domain and a path-planning instance, i.e., the planning domain $\mathcal{D} = \langle \mathcal{A}, L(\mathcal{A}), F_1(\mathcal{A}), \dots, F_k(\mathcal{A}) \rangle$ and the planning instance $\mathcal{P} = \langle \mathcal{D}, I(\mathcal{A}), G(\mathcal{A}) \rangle$. Call this \mathcal{P} the *planning instance induced by the given planning data*, or simply the *induced planning instance*.

Definition 1. *The path-planning problem asks, given as input the planning data $k, F_1, \dots, F_k, L, I, G, \mathcal{A}$, whether the induced path-planning instance has a solution. The restricted path-planning problem fixes $k, F_1, \dots, F_k, L, I, G$ and asks, given an arena \mathcal{A} as input, whether or not the induced path-planning instance has a solution.*

Remark 2. In order to talk about the complexity of this decision problem (and also of the notion of a tractable operator), we should specify how these objects are coded. We use any natural encoding. E.g., sets V are identified with sets of the form $\{1, 2, \dots, N\}$; the number of robots k is written in unary; relations (such as $E \subseteq V \times V$ and $P \subseteq V$) are coded by listing their elements; and a tractable operator is coded as the state diagram of a PTIME Turing machine (in Remark 4 we will see how to express operators as formulas rather than machines). The *size* of the planning data is the number of bits of its encoding. Note that the size of an arena $\mathcal{A} = \langle V, E \rangle$ is $O(|V|^2)$, and the size of a k -configuration over \mathcal{A} is $O(|V|^2 + |V|k)$.

Theorem 1. *The path-planning problem can be solved in PSPACE, or time exponential in $|V|$ and k .*

Proof. The number of configurations is $O(2^{|V|k})$, and thus the brute force algorithm takes EXPTIME in $|V|$ and k . However, since each configuration can be written in polynomial space (in the size of V and k), one can search the reachable configuration space using a nondeterministic polynomial-space algorithm. That is, the algorithm stores the current configuration c on its tape, nondeterministically guesses (in PTIME) a legal configuration d (and writes it on its tape), and then verifies that $c \vdash d$ (by Remark 1 the \vdash relation is computable in PTIME). The algorithm begins by guessing (in PTIME) a configuration in $I(\mathcal{A})$ and proceeds until the current configuration is in $G(\mathcal{A})$. This algorithm halts if and only if the induced planning problem has a solution. Now use the fact that NPSPACE = PSPACE. \square

Corollary 1. *For every $k, F_1, \dots, F_k, L, I, G$ the restricted path-planning problem can be solved in PSPACE, or time exponential in $|V|$.*

Since there are motion-planning problems that are PSPACE-hard (e.g., [12, 8]), the path-planning problem is also PSPACE-hard. For instance, *generalised rush-hour* is a generalisation of a children’s puzzle in which the objective is to slide cars in a grid-environment in order for a target car to reach a certain exit co-ordinate. Solving generalised rush-hour is PSPACE-complete [8].

Theorem 2. *The path-planning problem is PSPACE-hard.*

Proof. We describe a reduction from generalised rush-hour (GRH) to path-planning problems. Note that although [8] prove PSPACE-hardness when the GRH instances consist of cars (1×2 vehicles) and trucks (1×3 vehicles), it is known that using only cars suffice [29]. An instance of GRH consists of the width w and height h of the grid, a number n of cars, and for each car an orientation (horizontal or vertical) and co-ordinates of the head of each car (x_i, y_i) . We assume that the first car is the designated car, and that it has horizontal orientation. Each car can only move, forwards or backwards, in the direction of its orientation. The goal is to move the cars, one at a time, until the head of the designated car reaches the right-hand-side of the grid.¹

The main problem we have to solve is that in GRH one car moves at a time, while in our path-planning problems agents move concurrently. We solve this problem by treating the set of all cars as a single agent, i.e., let $k = 1$. This introduces the problem that we need to be able to distinguish between different cars. This is solved by placing each car on a disjoint copy of the grid. That is, a GRH configuration is encoded by the arena $w \times h \times n$ grid so that if the head of the i th car is (currently) at co-ordinate (a, b) then: if the car is horizontal then this car is coded by the vertices $a \times b \times i$ and $(a + 1) \times b \times i$, and if the car is vertical then this car is coded by the vertices $a \times b \times i$ and $a \times (b + 1) \times i$.

More precisely, given an instance of GRH define planning-data as follows. Let $k = 1$, let the arena be the $w \times h \times n$ grid, formally it has $V = [w] \times [h] \times [n]$ and $E = \{((x, y, z), (x', y', z')) : |x - x'| = 1 \text{ xor } |y - y'| = 1 \text{ xor } |z - z'| = 1\}$. We only describe how the operators map grid arenas of the form $a \times b \times c$ for $a, b, c \in \mathbb{N}$ (on other arenas the maps may be arbitrary). The legal configuration operator \mathcal{L} maps the grid arena $a \times b \times c$ to the set of configurations in which there is exactly one car on each of the c -levels; the \mathcal{G} operator maps the grid arena $a \times b \times c$ to the set of legal configurations in which the head of the first car has y -coordinate equal to b ; the \mathcal{I} operator maps the grid arena $w \times h \times n$ to the configuration encoding the initial layout of the GRH; the mobility operator maps the grid arena $a \times b \times c$ to the relation over $V = [a] \times [b] \times [c]$ that relates X to Y if the only difference between X and Y is that for some co-ordinate $i \leq c$, every element in $X \cap [a] \times [b] \times [i]$ moves one coordinate forwards in the

¹ The original definition of GRH unnecessarily allows any of the cars to be the target, the exit to be anywhere on the perimeter, and the target car to be horizontal or vertical [8].

direction of the orientation or every element moves one co-ordinate backwards in the direction of orientation. This completes the description of the reduction. It is not hard to see that the GRH has a solution if and only if the planning instance induced by the constructed data has a solution. \square

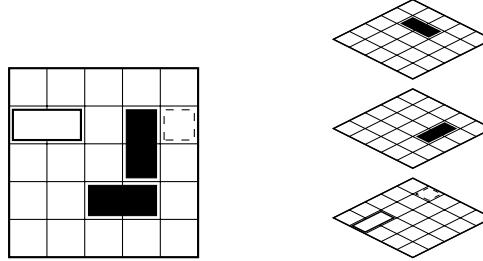


Fig. 2. A representation of the encoding for Generalised Rush Hour (GRH) to path-planning problem. On the left, an instance of GRH. The white rectangle represents the position of the designated car, the black rectangles represent the position of the other cars, the dashed square represents the exit point. On the right, the path-planning representation.

It is not clear if there exists a restricted path-planning problem that is PSPACE-hard. We remark that the proof just given does not work since a reduction would have to map an instance of generalised rush-hour (which includes the initial layout) to an arena \mathcal{A} (which does not include the initial configuration). We do know the following:

Proposition 1. *There exists $k, F_1, \dots, F_k, L, I, G$ such that the restricted path-planning problem is NP-hard.*

Proof. We reduce from the NP-complete problem that asks if a given vertex colouring of a graph by 3 colours is a proper colouring, i.e., that no two adjacent vertices have the same colour. The idea is to use $k = 3$ agents, the initial configurations are arbitrary colourings (i.e., I maps \mathcal{A} to $\langle \mathcal{A}, P_1, P_2, P_3 \rangle$ such that $P_i \cap P_j = \emptyset$ for $i \neq j$), the mobility relations map A to the identity relation on 2^V (i.e., agents do not move), every configuration is a legal configuration (i.e., L maps A to the set of all 3-configurations), and the goal configurations are proper colourings (i.e., G maps \mathcal{A} to $\langle \mathcal{A}, P_1, P_2, P_3 \rangle$ such that $(x, y) \in E$ and $x \in P_i, y \in P_j$ implies $i \neq j$). It is immediate that the reduction that maps \mathcal{A} to the data $k = 3, F_1, F_2, F_3, L, I, G$ is computable in PTIME, and that \mathcal{A} is 3-colourable if and only if the constructed restricted path-planning problem has a solution. \square

We now identify a general subclass of path-planning problems that are solvable in PTIME. We use the following definitions.

Definition 2. A mobility relation Δ is called:

- increasing if $(X, Y) \in \Delta \Rightarrow X \subseteq Y$,
- decreasing if $(X, Y) \in \Delta \Rightarrow Y \subseteq X$,
- monotone if it is increasing or decreasing,
- size-decreasing if $(X, Y) \in \Delta \Rightarrow |X| \geq |Y|$.

A mobility operator F is called increasing (resp. decreasing, monotone, size-preserving) if every mobility relation $F(\mathcal{A})$ is increasing (resp. decreasing, monotone, size-preserving).

A configuration operator C is called size-bounded if there is a constant $B \in \mathbb{N}$ such that for every arena \mathcal{A} , for every $i \leq k$, the cardinality of P_i in every configuration in $C(\mathcal{A})$ is at most B .

Note that in Examples 1 and 2, the mobility operators of the players are size-preserving, the mobility operator of the flood is increasing, and the initial-configuration operators are size-bounded (for all the agents, including, although we don't need this fact, the flood).

The following theorem should be contrasted with the fact that Theorem 1 implies that the time-complexity of restricted path-planning problems is exponential in $|V|$.

Theorem 3. Fix $B \in \mathbb{N}$, and consider the planning problem in which one restricts the input $\langle k, L, I, G, F_1, \dots, F_k \rangle$ so that I is size-bounded (by B), and each mobility operator F_i is monotone or size-decreasing. The time complexity of the restricted planning problem is polynomial in $|V|$.

Proof. The number N of reachable configurations is bounded above by a polynomial in $|V|$. Indeed: $N \leq \prod_{i \leq k} m_i |V|^{n_i}$ where $m_i = 1, n_i = 1$ if the mobility operator for agent i is monotone, and $m_i = 2, n_i = B$ if the mobility operator for agent i is size-preserving (since the number of subsets of V of size at most B is bounded above by $2|V|^B$). In particular, the product is $|V|^{O(k)}$, i.e., polynomial in $|V|$ and exponential in k . Now, since the successor configuration relation \vdash is computable in PTIME (see Remark 1) we can build the reachable configuration space in PTIME. Since the initial-configuration and goal-configuration operators are tractable, we can compute the set of initial configurations and the set of goal configurations in PTIME. Now we test if there is a path from some initial configuration to some goal configuration, which can be done in time quadratic in N . \square

Remark 3. Thus, the planning instances from Examples 1 and 2 are solvable in PTIME. The algorithm in the proof of Theorem 3 shows that Example 1 is solvable in time $O(|V|^2)$. With k people and one flood the algorithm runs in time $O(|V|^{k+1})$.

Remark 4. The reader who prefers logical specification languages (i.e., FOL) to computational ones (i.e., Turing Machines) may express their operators as formulas. Indeed, since the program-complexity of FOL (i.e., the complexity of model-checking FOL on finite structures in which the formula is fixed and the structure varies) is in PTIME, deduce that every FOL-definable operator is tractable.

Informally, the descriptions of the operators in Examples 1 and 2 can be expressed as formulas of FOL in appropriate signatures.

More precisely, fix unary predicate symbols P_1, \dots, P_k, A, B and a binary predicate symbol E (we use standard notation, e.g., [7]). We assume we have symbols for equality and set containment with their usual interpretations. A mobility operator F is *FOL-definable* if there is a first-order formula ϕ in the signature $S = (E, A, B)$, such that for every S -structure $\mathcal{M} = (V^{\mathcal{M}}, E^{\mathcal{M}}, A^{\mathcal{M}}, B^{\mathcal{M}})$ we have that $\mathcal{M} \models \phi$ if and only if $(A^{\mathcal{M}}, B^{\mathcal{M}}) \in F((V^{\mathcal{M}}, E^{\mathcal{M}}))$. For instance, the mobility operator for the flood in Example 1 is definable using a FOL-formula that states that A and B are singletons, say $A = \{a\}, B = \{b\}$, and that $(a, b) \in E$, e.g.,

$$\exists x \in A. \exists y \in B. \forall z. (z \in A \rightarrow x = z) \wedge (z \in B \rightarrow y = z) \wedge (x, y) \in E.$$

Similarly, a configuration operator C is *FOL-definable* if there is a first-order formula ϕ in the signature (E, P_1, \dots, P_k) such that for every S -structure $\mathcal{M} = (V^{\mathcal{M}}, E^{\mathcal{M}}, P_1^{\mathcal{M}}, \dots, P_k^{\mathcal{M}})$ we have that $\mathcal{M} \models \phi$ if and only if $\mathcal{M} \in C((V^{\mathcal{M}}, E^{\mathcal{M}}))$. For instance, a simple variation of the legal configurations in Example 2 is definable by the FOL-formula $\bigwedge_{i \neq j} \neg \exists x. x \in P_i \wedge x \in P_j$.

4 Comparison with other representations

The goal of this section is to analyse the translations of the abstract path-planning problems considered in this paper into standard representations of planning problems, i.e., the set-theoretical representation and the classical representation [10]. We illustrate by encoding the flood problem of Example 1.

A planning problem in the set-theoretical representation is a tuple $\mathcal{S} = \langle P, A, I, G \rangle$ where:

- P is a finite set of *atoms*,
- $I \subset P$ is the *initial state*,
- $G \subset P$ is the set of *goal propositions*,
- A is a set of *actions*, each action a is a tuple $\langle pre^+(a), pre^-(a), add(a), del(a) \rangle$ of subsets of P .

A *state* is a subset of P . A *plan* is any sequence (possibly empty sequence $\langle \rangle$) of actions $\pi = \langle a_1, a_2, \dots, a_N \rangle$. The state $a(s)$ produced by applying action a to state s is defined as follows: if $pre^+(a) \subseteq s$ and $pre^-(a) \cap s = \emptyset$, then $s' := (s \cup add(a)) \setminus del(a)$, and otherwise $s' := s$. The state produced by applying a plan π to a state s is defined inductively: $\langle \rangle(s) := s$ and $\langle \pi, a \rangle(s) := a(\pi.s)$. A plan π is a *solution* if $G \subseteq \pi(I)$.

For simplicity of exposition we now encode the flood scenario of Example 1 into the set-theoretical representation. Let $\langle V, E \rangle$ be the underlying graph, $q, r \in V$ be the starting positions of the person and the flood, respectively, and $t \in V$ the exit. Every vertex v of the graph is associated with two atoms p_v, f_v . The meaning of p_v is that the person currently occupies vertex v , and the

meaning of f_v is that vertex v is flooded. The initial state is $\iota = \{p_q, f_r\}$. The goal G is the set $\{t_v\}$. For every triple $\langle v, w, F \rangle \in E \times 2^V$ with $v, w \notin F$, there is an action a defined as follows:

- $pre^+(a) = \{p_v\} \cup \{f_z : z \in F\}$ (if the person is at v and F is flooded...),
- $pre^-(a) = \{f_z : z \notin F\}$ (and nowhere else is flooded...),
- $rem(a) = \{p_v\}$ (then remove the person from ...),
- $add(a) = \{p_w\} \cup \{f_z : \exists x \in F. (x, z) \in E\}$ (and place the person at w and expand the flood).

Then the person can escape the flood if and only if the planning problem \mathcal{S} has a solution.

The translation into the classical representation (i.e., states are ground literals in a first-order relational signature) is similar. In brief, there is a constant v for each vertex, constants p and f for the agents, a relation $at(x, y)$ that says that agent x occupies vertex y , and for each triple $\langle v, w, F \rangle \in E \times 2^V$ there is an action with precondition $\{at(p, v)\} \cup \{at(f, z) : z \in F\} \cup \{at(f, z) : z \notin F\}$ and effects $\{at(p, w), \neg at(p, v)\} \cup \{at(f, z) : z \in E(F)\}$.

We now present a simple analysis. First, the size of both of the representations is exponential in $|V|$. In contrast, the size of the representation in Example 1 is polynomial in $|V|$. Second, since the actions of the agents are concurrent, none of the actions are monotone, e.g., it is note the case that once a tuple is removed from at (or a variable in the set-theoretic representation is removed) it is not added later. In contrast, our representation neatly separates the abilities of the multiple agents (i.e., some are monotone, others are not) even though the agents behave concurrently. Third, consider the *interaction graph* for the set-theoretic representation [3]: its vertices are the atoms, and there is an edge from atom x to atom y if there is some action a such that $x \in pre^+(a) \cup pre^-(a) \cup rem(a) \cup add(a)$ and $y \in rem(a) \cup add(a)$. The main result in [3] implies that if there is no bound on the size of the strong connected components of the interaction graph then the planning problem (i.e., the existence problem) is not in PTIME unless an established complexity-theoretic assumption fails. Note that the arena $\mathcal{A} = \langle V, E \rangle$ embeds in the interaction graph. Thus there is no bound on the size of the strongly connected components of the interaction graphs, even in the flood example. In contrast, since our framework is tailored for path-planning problems (and not general planning problems), we were able to exhibit PTIME algorithms for such path-planning problems on arbitrary arenas (Theorem 3).

5 Summary and Future Work

We exhibited a natural representation of path-planning problems in which the arena and the positions of the agents in the arena are coded in a natural graph-theoretic way, rather than as an evaluation of Boolean propositions (as in, e.g., the set-theoretical representation). Our formalism can represent static environments (e.g., fixed obstacles are coded by the lack of an edge in the graph) and

dynamic but predictable environments (e.g., flooding). We gave a PSPACE algorithm for solving the planning problem in these settings, which is also solvable in time exponential in the number of robots k and the size of the arena $|V|$.

By restricting the abilities of the agents (to be monotone or size-preserving), we showed that the planning problem can be solved in time polynomial in the size of the arena, i.e., for a fixed number of agents k but a varying arena, these restricted planning problems are PTIME in the size of the arena $|V|$. The same ideas can also be used to get PTIME algorithms for b -bounded piecewise monotone mobility relations, e.g., for $b = 1$ the flood starts spreading, but at some point in time stops spreading and begins to recede. The reason is that the size of the configuration space blows up, for each b -bounded piecewise monotone agent, by a multiplicative factor of $|V|^b$.

The main open question in this work is the exact complexity of restricted planning-problems, i.e., is there a restricted planning-problem that is PSPACE-hard? We know that there is a restricted planning-problem with three agents that is NP-hard (Proposition 1).

This paper opens up many avenues for extending the model and studying the complexity: optimal planning, e.g., there are costs associated with moving that need to be minimised [19] — our paper only deals with feasible planning, i.e., “does there exist a plan”; adversarial agents that are non-deterministic or probabilistic [14, 17, 6, 9, 22] — our paper only covers, implicitly, deterministic adversaries, such as the flood; and more expressive goals, such as those expressible in temporal logics, [5, 24] or quantitative goals — our paper only deals with attainment/reachability goals; imperfect information, e.g., one does not know the exact starting position of the flood, or one does not know the speed of the flood.

References

1. R. I. Brafman and C. Domshlak. From one to many: Planning for loosely coupled multi-agent systems. In J. Rintanen, B. Nebel, J. C. Beck, and E. A. Hansen, editors, *ICAPS*, pages 28–35. AAAI, 2008.
2. T. Bylander. The computational complexity of propositional strips planning. *Artificial Intelligence*, 69(1):165–204, 1994.
3. H. Chen and O. Giménez. Causal graphs and structurally restricted planning. *J. Comput. Syst. Sci.*, 76(7):579–592, 2010.
4. M. C. Cooper, F. Maris, and P. Régnier. Monotone temporal planning: Tractability, extensions and applications. *J. Artif. Intell. Res. (JAIR)*, 50:447–485, 2014.
5. G. De Giacomo and M. Y. Vardi. Automata-theoretic approach to planning for temporally extended goals. In S. Biundo and M. Fox, editors, *Recent Advances in AI Planning, ECP*, volume 1809 of *LNCS*, pages 226–238. Springer, 1999.
6. T. L. Dean, R. Givan, and K. Kim. Solving stochastic planning problems with large state and action spaces. In R. G. Simmons, M. M. Veloso, and S. F. Smith, editors, *AIPS*, pages 102–110. AAAI, 1998.
7. H. Enderton. *A mathematical introduction to logic*. Academic Press, 1972.
8. G. W. Flake and E. B. Baum. Rush hour is pspace-complete, or “why you should generously tip parking lot attendants”. *Theoretical Computer Science*, 270(1-2):895 – 911, 2002.

9. H. Geffner and B. Bonet. *A Concise Introduction to Models and Methods for Automated Planning*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool, 2013.
10. M. Ghallab, D. S. Nau, and P. Traverso. *Automated planning - theory and practice*. Elsevier, 2004.
11. F. Giunchiglia and P. Traverso. Planning as model checking. In *Recent Advances in AI Planning*, pages 1–20, 1999.
12. J. Hopcroft, J. Schwartz, and M. Sharir. On the complexity of motion planning for multiple independent objects; PSPACE-hardness of the “warehouseman’s problem”. Technical report, Courant Institute of Mathematical Sciences, New York, 1984.
13. W. Jamroga. Strategic planning through model checking of ATL formulae. In *ICAISC*, volume 3070 of *Lecture Notes in Computer Science*, pages 879–884. Springer, 2004.
14. L. P. Kaelbling, M. L. Littman, and A. R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial intelligence*, 101(1):99–134, 1998.
15. D. Kornhauser, G. Miller, and P. Spirakis. Coordinating pebble motion on graphs, the diameter of permutation groups, and applications. In *FOCS*, pages 241–250. IEEE, 1984.
16. A. Krontiris, Q. Sajid, and K. Bekris. Towards using discrete multiagent pathfinding to address continuous problems. In *Proc. AAAI Workshop on Multiagent Pathfinding*, 2012.
17. M. Lamiri, X. Xie, A. Dolgui, and F. Grimaud. A stochastic model for operating room planning with elective and emergency demand for surgery. *European Journal of Operational Research*, 185(3):1026–1037, 2008.
18. J.-C. Latombe. *Robot motion planning*, volume 124 of *International Series in Engineering and Computer Science*. Springer, 2012.
19. S. M. LaValle. *Planning Algorithms*. Cambridge University Press, 2006.
20. V. J. Lumelsky, A. Stepanov, et al. Dynamic path planning for a mobile automaton with limited information on the environment. *Automatic Control, IEEE Transactions on*, 31(11):1058–1063, 1986.
21. F. Mogavero, A. Murano, and M. Y. Vardi. Relentful strategic reasoning in alternating-time temporal logic. *Journal of Logic and Computation*, to appear, 2014.
22. A. Murano and L. Sorrentino. A game-based model for human-robots interaction. In *Workshop "From Objects to Agents" (WOA)*, volume 1382 of *CEUR Workshop Proceedings*, pages 146–150. CEUR-WS.org, 2015.
23. C. H. Papadimitriou, P. Raghavan, M. Sudan, and H. Tamaki. Motion planning on a graph (extended abstract). In *FOCS*, pages 511–520. IEEE, 1994.
24. M. Pistore and P. Traverso. Planning as model checking for extended goals in non-deterministic domains. In T. Walsh, editor, *IJCAI*, pages 479–486. IJCAI/AAAI, 2001.
25. M. Pistore and M. Y. Vardi. The planning spectrum - one, two, three, infinity. *J. Artif. Intell. Res. (JAIR)*, 30:101–132, 2007.
26. G. Röger and M. Helmert. Non-optimal multi-agent pathfinding is solved (since 1984). In D. Borrajo, A. Felner, R. E. Korf, M. Likhachev, C. L. López, W. Ruml, and N. R. Sturtevant, editors, *SOCS*. AAAI, 2012.
27. O. Souissi, R. Benatallah, D. Duvivier, A. Artiba, N. Belanger, and P. Feyzeau. Path planning: A 2013 survey. In *Industrial Engineering and Systems Management (IESM)*, pages 1–8, Oct 2013.

28. P. Surynek. An application of pebble motion on graphs to abstract multi-robot path planning. In *ICTAI*, pages 151–158. IEEE, 2009.
29. J. Tromp and R. Cilibrasi. Limits of Rush Hour Logic Complexity. *CoRR*, abs/cs/0502068, 2005.
30. J. van den Berg, J. Snoeyink, M. C. Lin, and D. Manocha. Centralized path planning for multiple robots: Optimal decoupling into sequential plans. In *Robotics: Science and Systems V*, 2009.
31. W. van der Hoek and M. Wooldridge. Tractable multiagent planning for epistemic goals. In *AAMAS*, pages 1167–1174. ACM, 2002.
32. G. T. Wilfong. Motion planning in the presence of movable obstacles. *Ann. Math. Artif. Intell.*, 3(1):131–150, 1991.

Parameterised Verification of Autonomous Mobile-Agents in Static but Unknown Environments

Sasha Rubin^{*}

Università degli Studi di Napoli “Federico II”

Naples, Italy

sasha.rubin@gmail.com

ABSTRACT

Automata walking on graphs are a mathematical formalisation of autonomous mobile agents with limited memory operating in discrete environments. This paper establishes a framework in which to model and automatically verify that autonomous mobile agents correctly perform their tasks. The framework consists of a logical language tailored for expressing agent tasks, and an algorithm solving the *parameterised* verification problem, where the graphs are treated as the parameter. We reduce the parameterised verification problem to classic questions in automata theory and monadic second order logic, i.e., universality and validity problems.

We illustrate the framework by instantiating it to a popular model of robot from the distributed computing literature.

This work clarifies the border between classes of mobile-agent systems that have decidable parameterised verification problem, and those that do not.

Categories and Subject Descriptors

D.2.4 [Software/Program Verification]: Formal Methods; F.1.1 [Computation by Abstract Devices]: Models of Computation—*Automata*; C.2.2 [Computer Communication Networks]: Network Protocols—*Protocol Verification*; C.2.4 [Computer Communication Networks]: Distributed Systems; I.2.11 [Artificial Intelligence]: Distributed Artificial Intelligence—*Multiagent systems*

Keywords

Model Checking; Logic; Automata Theory; Distributed Networks; Autonomous Mobile Agents; Robots; Parameterised Verification

1. INTRODUCTION

Autonomous mobile agents are designed to achieve some task in an environment, e.g., exploration, or rendezvous. They are in an *unknown* environment if they do not have

*Marie Curie fellow of the Istituto Nazionale di Alta Matematica.

Appears in: *Proceedings of the 14th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2015), Bordini, Elkind, Weiss, Yolum (eds.), May 4–8, 2015, Istanbul, Turkey.*

Copyright © 2015, International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org). All rights reserved.

global information about the environment, e.g., mobile software exploring hostile computer networks, or physical robots that rendezvous in an environment not reachable by humans. This paper studies a mathematical formalisation of autonomous mobile agents in discrete environments, i.e., robots on finite graphs.

The distributed computing community has recently proposed and studied a number of models of robot systems [5, 33, 28, 18, 21, 16, 27]. Theorems from this literature are *parameterised* i.e., they may involve graph-parameters (e.g., the maximum degree, the number of vertices, the diameter), memory parameters (e.g., the number of internal states of the robot protocol), and the number of robots may be a parameter.

However, until recently [31, 4, 35, 32] there has been little emphasis on formal analysis of correctness of robots in a parameterised setting. In this paper we apply formal methods to the parameterised verification problem in which it is the *environment that is parameterised*. This is how we model that robots operate in unknown environments. Formally, we address the following decision problem.

Parameterised Verification Problem: Given robots \bar{R} , decide if they solve the task T on all graphs $G \in \mathcal{G}$.

In contrast, the classic (non-parameterised) verification problem states:

Verification Problem: Given robots \bar{R} and a graph $G \in \mathcal{G}$, decide if robots \bar{R} solve the task T on G.

The non-parameterised verification problem is often handled using model-checking [14]. The parameterised verification problem corresponds to a family of model-checking problems (one for each $G \in \mathcal{G}$), or equivalently, a single infinite-state model-checking problem. A variety of techniques have been applied to such systems, fully discussed in Section 7 (Related Work), however none seem suited to the problem in which the environment is parameterised. Fortunately, the theory of automata and logic has produced a rich understanding of the expressive power of certain logics, e.g., monadic second-order logic **MSOL**, over certain classes of graphs, e.g., context-free sets of graphs [44, 45, 29, 17]. Our framework reduces the parameterised verification problem to classic questions in automata theory and **MSOL**, i.e., universality and validity problems. The key observations are that: i) robots are graph-walking automata, and can be defined in **MSOL**, and ii) robot tasks correspond to properties of runs of automata which are often definable in **MSOL**.

Modeling Choices.

There are a number of different modeling choices for mo-

bile agents [38]. We list them, and emphasise (using underlining) the choices made in this paper: (i) is the environment continuous (e.g., the plane) or discrete (e.g. a finite graph whose edges are labeled by directions or port numbers)? (ii) do agents act synchronously or asynchronously? (iii) are agents probabilistic or non-deterministic? (iv) is there one agent or are there multiple agents, and are the number of agents known or unknown? (v) is the environment static, or can agents affect their environment (e.g., by marking the nodes of a graph)? Moreover, (vi) how much information about the environment is known, a priori, to the agents? We assume agents do not have global knowledge of the environment, and in particular the size is not known and nodes in the environment do not have unique identifiers. And finally, (vii) how do agents communicate and sense their environment? We assume agents can *sense their positions in the graph* as well as those of the other agents (in the case of multiple agents), i.e., robots acquire information about the current state of the environment solely by vision (we use logical formulas, which we call tests, to define these sensing abilities).

Aim and Contributions.

The main aim of this work is to provide a clearer understanding of the border between classes of autonomous mobile-agent systems that have decidable parameterised verification problem, and those that do not.

In particular, we provide a quite general formalisation of robot systems (using the modeling choices above) in which all components (i.e., environments, robots, and tasks) are modeled separately. We formalise reasoning about robot systems as the parameterised verification problem (PVP) where the environment is treated as a parameter. We show how to reduce the PVP to problems about automata and logic, i.e., universality and validity problems. We then prove that the PVP is decidable for suitable restrictions of the components, notably the case of a single robot on context-free sets of graphs. We illustrate the power of the framework by instantiating it to a model of robot systems from the distributed computing literature. This instantiation falls into our decidable restrictions.

2. BACKGROUND: AUTOMATA THEORY

In order to make this paper self-contained, this section contains the necessary background. In particular, we will use basic notions from mathematical logic and automata theory to formalise the robot systems (a full logical background can be found in [22], and a smaller discussion of monadic second-order logic **MSOL** over graphs can be found in [17, Section 1.3.1]), and easy-to-state theorems (e.g., Kleene’s Theorem about the equivalence of regular expressions and automata, and Courcelle’s Theorem on satisfiability of **MSOL** over context-free sets of graphs [17]) to give an algorithm for solving the parameterised verification problem.

Write B^ω for the set of infinite sequences over alphabet B , and write B^* for the set of finite sequences. The empty sequence is denoted ϵ . Write $[n]$ for the set $\{1, 2, \dots, n\}$.

Graphs.

A Σ -graph, or *graph*, G , is a tuple (V, E, Σ, λ) where V is a finite set of vertices, $E \subseteq V \times V$ is a relation called the *edge relation*, Σ is a finite set of *edge labels*, and $\lambda : E \rightarrow \Sigma$ is the

edge labeling function. The *out-degree* of a vertex v , written $\deg(v)$, is the cardinality of the set $\{(v, w) \in E : w \in V\}$ of outgoing edges of v .

Sometimes the edge labels give a sense of direction:

EXAMPLE 1. An *L-labeled grid* is a graph with $V = [n] \times [m]$, $\Sigma = L \times \{N, S, E, W\}$, and labels $\lambda((x, y), (x+1, y)) \in L \times \{E\}$, etc., where L is a finite set and $n, m \in \mathbb{N}$. An *L-labeled line* is an *L-labeled grid* with $V = [n] \times [1]$. If $|L| = 1$ then we call the grid (or line) unlabeled.

EXAMPLE 2. A Δ -ary tree (for $\Delta \in \mathbb{N}$) is a Σ -graph (V, E, Σ, λ) where (V, E) is a tree, $\Sigma = [\Delta] \cup \{\text{up}\}$, and λ labels the edge leading to the node in direction i (if it exists) by i , and the edge leading to the parent of a node (other than the root) is labelled by up. We may rename the labels for convenience, e.g., for binary trees ($\Delta = 2$) we let $\Sigma = \{\text{lc}, \text{rc}, \text{up}\}$ where lc replaces 1 and rc replaces 2.

First-order and Monadic Second-order Logic.

Formulas will be interpreted in Σ -graphs G (an exception are formulas in Section 4.3). Define the set of monadic second-order formulas **MSOL**(Σ) as follows. Formulas of **MSOL**(Σ) are built using *first-order variables* x, y, \dots that vary over vertices, and *set variables* X, Y, \dots that vary over sets of vertices. The *atomic formulas* (when interpreted over Σ -graphs) are: $x = y$ (denoting that vertex x is the same as vertex y), $x \in X$ (denoting that vertex x is in the set of vertices X), and $\text{edg}_\sigma(x, y)$ (denoting that there is an edge from x to y labeled $\sigma \in \Sigma$). The formulas of **MSOL**(Σ) are built from the atomic formulas using the Boolean connectives (i.e., $\neg, \vee, \wedge, \rightarrow$) and variable quantification (i.e., \forall, \exists over both types of variables). The fragment of **MSOL**(Σ) which does not mention set variables is called *first-order logic*, denoted **FOL**(Σ). Write **MSOL** $_k$ (Σ) for formulas with k -many free variables.

Here are some examples of formulas and their meanings:

- MF1. The formula $\forall x(x \in X \rightarrow x \in Y)$ means that $X \subseteq Y$. Similarly, there are formulas for the set operations $\cup, \cap, =$, and relative complement $X \setminus Y$.
- MF2. The formula $\text{edg}(x, y) := \bigvee_{\sigma \in \Sigma} \text{edg}_\sigma(x, y)$ means that there is an edge from x to y (here Σ is assumed to be finite).
- MF3. The formula $\exists x \exists y(x \neq y \wedge \text{edg}(z, x) \wedge \text{edg}(z, y))$ means that $\deg(z) \geq 2$.
- MF4. If $\phi(x, y)$ is an **MSOL**(Σ) formula then define $\phi^*(x, y)$ by $\forall Z[(\text{closed}_\phi(Z) \wedge x \in Z) \rightarrow y \in Z]$. Here $\text{closed}_\phi(Z)$ is defined as $\forall a \forall b[(a \in Z \wedge \phi(a, b)) \rightarrow b \in Z]$. Note that ϕ^* is an **MSOL**(Σ)-formula.

Note that $\phi^*(x, y)$ holds in a graph G if and only if there exists a finite sequence of vertices $v_1 v_2 \dots v_m \in V^*$ (here $m \geq 1$) such that $x = v_1, y = v_m$ and $\phi(v_i, v_{i+1})$ holds in G for all $i < m$. This shows that if a binary relation is **MSOL**-definable, then so is its transitive-closure.

Similarly, define $\phi^\omega(x, y) := \phi^*(x, y) \wedge \exists z(\phi(y, z) \wedge \phi^*(z, y))$. Note $\phi^\omega(x, y)$ holds in a graph G if and only if there is an infinite sequence of vertices $v_1 v_2 \dots \in V^\omega$ with $x = v_1, v_i = y$ for infinitely many $i \in \mathbb{N}$, and $\phi(v_i, v_{i+1})$ holds in G for all $i \in \mathbb{N}$. This uses the fact that V is finite.

MF5. The *k*-ary transitive-closure operator is the function that maps a *2k*-ary relation $\rho(\bar{x}, \bar{y})$ to the *2k*-ary relation $\rho^*(\bar{x}, \bar{y})$ such that, in every graph G : $\rho^*(\bar{x}, \bar{y})$ holds if and only if there exists a sequence $\bar{v}_1, \bar{v}_2, \dots, \bar{v}_m$ such that $\bar{x} = \bar{v}_1$, $\bar{y} = \bar{v}_m$, and $\rho(\bar{v}_i, \bar{v}_{i+1})$ holds for every $i < m$. For $k > 1$, it is not the case that if a *k*-ary relation ϕ is MSOL-definable, then so is its *k*-ary transitive-closure, because, intuitively, this would require having *k*-ary relation variables Z .¹

For a formula $\phi(x_1, \dots, x_k)$, a graph G , and $v_1, \dots, v_k \in V$, write $G \models \phi(v_1, \dots, v_k)$ to mean that ϕ holds in G with variable x_i substituted by vertex v_i (for $i \leq k$).

The Validity Problem and Courcelle's Theorem.

A *sentence* is a formula with no free variables. Let Φ be a set of sentences, and let \mathcal{G} be a set of graphs. The *Φ -validity problem* of \mathcal{G} is to decide, given $\phi \in \Phi$, whether or not for all graphs $G \in \mathcal{G}$, it holds that $G \models \phi$. Unfortunately for us, the $\text{FOL}(\Sigma)$ -validity problem for \mathcal{G} = the set of all Σ -graphs is undecidable [22]. On the other hand, many interesting cases have decidable validity. One version of Courcelle's Theorem states that the MSOL-validity problem is decidable for every context-free set of graphs \mathcal{G} [17].² Context-free sets of graphs are the analogue of context-free sets of strings, and are generated by graph grammars or equations using certain graph operations. Examples include, for a fixed alphabet, the set of labeled lines, the set of rings, the set of trees, the set of series-parallel graphs, the set of cliques, but not the set of all grids.

Automata and Regular Expressions.

Ordinary *regular-expressions* over a finite alphabet B are built from the sets \emptyset , $\{\epsilon\}$, and $\{b\}$ for $b \in B$, and the operations union $+$, concatenation \cdot , and Kleene-star * . Kleene's Theorem states that the languages recognised by regular expressions over alphabet B are exactly those recognised by finite automata over alphabet B .

An ω -*regular expression* over the finite alphabet B is inductively defined to be of the form: \exp^ω , $\exp \cdot r$, or $r + r'$, where \exp is an ordinary regular-expression over B , and r, r' are ω -regular expressions over B . An ω -regular language is one defined by an ω -regular expression. A variation of Kleene's Theorem says that the languages recognised by ω -regular expressions over alphabet B are exactly the languages recognised by Büchi automata over alphabet B (which are like non-deterministic finite automata except that they take infinite words as input, and accept if some accepting state occurs infinitely often).

3. THE MODEL OF ROBOT SYSTEMS

We provide a framework for modeling multi-robot systems parameterised by their environment.

¹To prove this formally, note that 2-ary transitive closure on finite words (i.e., binary-labeled lines) can define the non-regular language $\{0^n 1^n : n \in \mathbb{N}\}$; now use the fact that, over finite words, MSOL can only define the regular languages, part of a result known as the Büchi-Elgot-Trakhtenbrot Theorem, see [45].

²Actually, a strong form of the theorem states that there is an algorithm that given a description of a context-free set of graphs \mathcal{G} and an MSOL-formula ϕ decides if every graph in \mathcal{G} satisfies ϕ .

3.1 The model of robot systems

In this section we define robot systems, i.e., robot protocols, environments, and tasks.

Environments are modeled as Σ -graphs,³ and robots are modeled as regular languages of instructions, as in [7]. An instruction either tells the robot to move along an edge, or it allows the robot to test the positions of the robots (e.g., a robot can learn which other robots are at the same vertex as it is, or if there is a robot north of it). Tests are formalised as logical formulas. As discussed in Section 5, our model is general enough to be able to express a popular model of robot systems found in the distributed computing literature [27, 21, 16, 33, 28, 18].

We first define robot systems consisting of a single robot, and then define multi-robot systems.

Robot Instruction Set.

A robot follows instructions from the *instruction set*

$$\text{INS}_{\Sigma,1} := \{\uparrow_\sigma : \sigma \in \Sigma\} \cup \text{MSOL}_1(\Sigma).$$

Instructions are of two kinds, *moves* and *tests*:⁴ \uparrow_σ tells the robot to move from its current vertex along the edge labeled σ ; and $\text{MSOL}_1(\Sigma)$ consists of formulas $\tau(x)$ that allow the robot to test that $\tau(x)$ holds in G , where x is the current vertex of the robot in G . Typical tests include: is the degree of the current vertex at least d ? is there an edge out of the current vertex labeled σ ? Here is a more complex test: is there a path from the current vertex that only uses edges labeled σ to a vertex of degree d ?

For a sequence $\text{ins} \in (\text{INS}_{\Sigma,1})^*$, write $[[\text{ins}]]_G \subseteq V^2$ for the set of pairs of vertices (u, v) such that, in G , one can reach v from u by following the instructions ins . Formally,

1. $(u, v) \in [[\epsilon]]_G$ iff $u = v$,
2. $(u, v) \in [[\uparrow_\sigma]]_G$ iff $\lambda(u, v) = \sigma$,
3. $(u, v) \in [[\tau(x)]]_G$ iff $u = v$ and $G \models \tau(u)$, and
4. $(u, v) \in [[d \cdot e]]_G$ for $d \in (\text{INS}_{\Sigma,1})^*$ and $e \in \text{INS}_{\Sigma,1}$ iff there exists $z \in V$ such that $(u, z) \in [[d]]_G$ and $(z, v) \in [[e]]_G$.

Robot Protocols.

Fix a set of edge-labels Σ . A *1-robot*, or *robot*, R , is a pair (Q, δ) where Q is a finite set of *states*, and $\delta \subset Q \times \text{INS}_{\Sigma,1} \times Q$ is a finite *transition* relation.

We sometimes designate certain states of the robot to be *initial*, denoted $I \subseteq Q$ and certain states to be *repeating*, denoted $A \subseteq Q$. A state $p \in Q$ is called *halting* if the robot has the transition (p, true, p) , i.e., state p is a sink. Thus we model a halting robot as one that stays in the same state forever. The set of halting states is denoted $H \subseteq Q$.

EXAMPLE 3. The robot in Figure 1 does a (perpetual) depth-first search of trees in which every node has zero or two children.⁵ Tests are written *leaf*, *lc*, *rc*, *root* (whose meanings are “is the current node a leaf?”, “left-child?”, “right-child?”, “the root?”), and move instructions are written \uparrow_{up} ,

³The fact that Σ -graphs are finite corresponds to our modeling assumption that environments are *discrete*.

⁴Looking at the instruction set we see that robots cannot alter the graph, i.e., the environment is *static*.

⁵To make the diagram readable, an edge may be labeled by multiple successive actions separated by semicolons.

\uparrow_{lc} , and \uparrow_{rc} (whose meanings are “move up to the parent”, “to the left-child”, “to the right-child”).

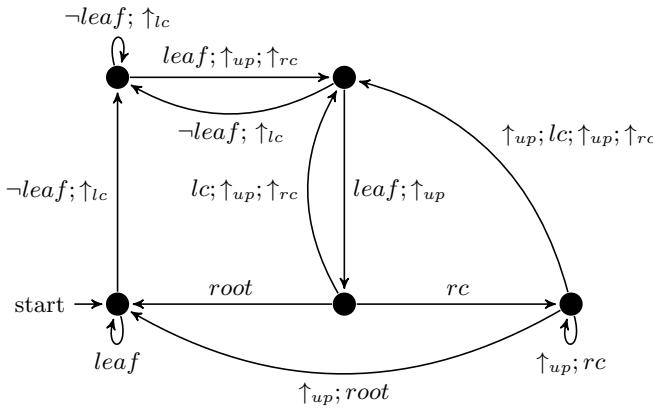


Figure 1: Robot that does a DFS on binary trees.

Configurations and Runs.

A robot walks about a Σ -graph G . Let $R = (Q, \delta)$ be a robot with instruction set $\text{INS}_{\Sigma, 1}$. A configuration c of R on G is a pair $\langle v, q \rangle \in V \times Q$. Say that the configuration $\langle v, q \rangle$ has position v . A configuration is *initial* (resp. *halting, repeating*) if q is. The following definition expresses that one configuration results from another after the robot executes a single instruction: for configurations $c = \langle v, p \rangle$ and $d = \langle w, q \rangle$, say that d *results from* c iff there is a transition (p, ins, q) of δ such that $\langle v, w \rangle \in [[ins]]_G$. A run α of R on G starting with an initial configuration c is an infinite sequence $c_1 c_2 c_3 \dots$ of configurations such that $c_1 = c$ and for all i , c_{i+1} results from c_i .

The set of positions of a run $\alpha = (v_1, q_1)(v_2, q_2) \dots$ is the set of positions $\{v_1, v_2, \dots\}$ of its configurations. The sequence of positions of a run α is the sequence of positions $v_1 v_2 \dots$ of its configurations.

Multi-Robot Systems.

We extend the previous definitions to k -many of robots.

A k -robot ensemble is a sequence $\langle R_1, \dots, R_k \rangle$ where, writing $R_i = (Q_i, \delta_i)$, each Q_i is a finite set of states, and each $\delta_i \subseteq Q_i \times \text{INS}_{\Sigma, k} \times Q_i$ is a finite transition relation, where the instruction set is $\text{INS}_{\Sigma, k} := \{\uparrow_\sigma : \sigma \in \Sigma\} \cup \text{MSOL}_k(\Sigma)$. For a sequence $ins \in ((\text{INS}_{\Sigma, k})^k)^*$ define⁶ $[[ins]]_G \subseteq V^{2k}$ such that $(\bar{u}, \bar{v}) \in [[ins]]_G$ if and only if, in G , one can reach \bar{v} from \bar{u} by following the instructions in ins .

Formally,

1. If $ins = \epsilon$, then $(\bar{u}, \bar{v}) \in [[ins]]_G$ if and only if $\bar{u} = \bar{v}$;
2. If $ins = (d_1, \dots, d_k) \in \text{INS}_{\Sigma, k}$ then $(\bar{u}, \bar{v}) \in [[ins]]_G$ if, for each $i \leq k$:
 - (a) if $d_i = \uparrow_\sigma$ then $\lambda(u_i, v_i) = \sigma$,
 - (b) if $d_i = \tau(x_1, \dots, x_k)$ then $u_i = v_i$ and $G \models \tau(u_1, \dots, u_k)$.

⁶To improve readability, the notation $[[\]]$ does not mention k .

3. If $ins = d \cdot e$ then $(\bar{u}, \bar{v}) \in [[ins]]_G$ if and only if there exists $\bar{z} \in V$ such that $(\bar{u}, \bar{z}) \in [[d]]_G$ and $(\bar{z}, \bar{v}) \in [[e]]_G$.

Fix a Σ -graph G . A configuration c of $\langle R_1, \dots, R_k \rangle$ on graph G is a pair $\langle \bar{v}, \bar{q} \rangle \in V^k \times \prod_{i \leq k} Q_i$. A configuration is *initial* if q_i is the initial state of robot R_i (for all $i \leq k$). The following definition expresses that one configuration results from another after the robots simultaneously execute their own next instruction (which may be to move along an edge labeled σ , or to test the current position of all the robots):⁷ configuration $\langle \bar{w}, \bar{q} \rangle$ results from $\langle \bar{v}, \bar{p} \rangle$ iff there are transitions $(p_i, ins_i, q_i) \in \delta_i$ (for $i \leq k$) such that $(\bar{v}, \bar{w}) \in [[(ins_1, \dots, ins_k)]]_G$. Runs and their sets of positions and sequences are defined as before.

Robot Tasks.

Robots should achieve some task in their environment: a k -robot task, or simply a task, T , is a function that maps a graph G to a set of sequences of positions of G , i.e., $T(G) \subseteq (V^k)^\omega$. A robot-ensemble \bar{R} achieves T on G if for every run α of \bar{R} on G it holds that $\alpha' \in T(G)$, where α' is the sequence of positions of the run α .

We give some examples of foundational robot tasks [34]:

- RT1. A robot explores and halts if, no matter where it starts, it a) eventually halts, and b) visits every vertex of the graph at least once.
- RT2. A robot perpetually explores if, no matter where it starts, it visits every vertex of G infinitely often.
- RT3. A robot explores and returns if, no matter where it starts, it a) eventually stops where it started, and b) visits every vertex of the graph at least once.
- RT4. An ensemble of robots collaboratively explores a graph if, no matter where they start, every node is eventually visited by at least one robot.
- RT5. A k -robot ensemble gathers if, no matter where each robot starts, there is a vertex z , such that eventually every robot is in z .

4. REASONING ABOUT ROBOT SYSTEMS

We formalise the parameterised verification problem for robot protocols, and then describe our solution to it (Theorem 2) which shows how to reduce parameterised verification in the case of a single robot to the logical validity problem of certain logics.

4.1 The Parameterised Verification Problem

The parameterised verification problem depends on a set of graphs \mathcal{G} , a set of robot ensembles \mathcal{R} , and a robot task T . Note that \mathcal{G} is typically infinite.

DEFINITION 1. The *parameterised verification problem* $\text{PVP}_T(\mathcal{G}, \mathcal{R})$ states: given a robot ensemble \bar{R} from \mathcal{R} , decide whether or not \bar{R} achieves the task T on every graph $G \in \mathcal{G}$.

Unfortunately, this problem is easily undecidable for the types of systems we have:

⁷This is where we model the assumption that robots act synchronously.

THEOREM 1. *There exists a task T and computable sets \mathcal{G} and \mathcal{R} such that $\text{PVP}_T(\mathcal{G}, \mathcal{R})$ is undecidable.*

In particular, we can choose $k = 1$, T to be the task “never halt”, \mathcal{G} to be the set of unlabeled-grids, and \mathcal{R} to be all robots; or we can choose $k = 2$, T to be the task “no robot ever halts”, \mathcal{G} to be the set of unlabeled-lines, and \mathcal{R} to be all robots.

PROOF SKETCH. Since the proof technique is a standard, we merely sketch it. We reduce the non-halting problem for (Turing-powerful) two-counter machines to $\text{PVP}_T(\mathcal{G}, \mathcal{R})$, i.e., given machine M build robot(s) \bar{R} such that M accepts no input if and only if \bar{R} achieves task T on all graphs in \mathcal{G} .

Case $k = 1$: As observed by [8] for their “2-dimensional automata”, one robot on a grid can simulate a two-counter machine: counter values $(n, m) \in \mathbb{N}^2$ are encoded by the robot being at position (n, m) of the grid. The only tests that are needed are to check whether or not the robot is on the boundary (this is to simulate the counter machine’s “test for zero”, as well as to alert about a counter overflow).

Case $k = 2$: The idea is that two distinguishable robots on a line can simulate one robot on a square grid; the position of the i th robot on the line gives the i th co-ordinate of the single robot on the grid. The robots only need to be able to test if they are at the end points of the line. Alternatively, one can directly code computations of Turing machines, as was done by [40] to show that universality of 2-head one-way finite-state automata is undecidable. \square

In light of this negative result, our main task is to understand in what way we can restrict the systems to get decidability.

4.2 Reducing Parameterised Verification to Validity

We first describe, at a high level, the approach we use to solve (restricted cases of) the parameterised verification problem $\text{PVP}_T(\mathcal{G}, \mathcal{R})$. Suppose we can build, for every k -ensemble \bar{R} of robots, a formula $\phi_{\bar{R}, T}$ such that for all graphs G the following are equivalent:

- $G \models \phi_{\bar{R}, T}$
- \bar{R} achieves task T on G .

Then, for every \mathcal{R} and \mathcal{G} , we would have reduced the parameterised verification problem $\text{PVP}_T(\mathcal{G}, \mathcal{R})$ to the $\Phi_{\bar{R}, T}$ -validity problem for \mathcal{G} where $\Phi_{\bar{R}, T}$ is the set of formulas $\{\phi_{\bar{R}, T} : \bar{R} \in \mathcal{R}\}$.⁸

We now detail this approach in the case of a single robot, i.e., $k = 1$.

LEMMA 1. *Let $R = (Q, \delta)$ be a robot over instruction set $\text{INS}_{\Sigma, 1}$, and let $p, q \in Q$.*

We can build $\text{MSOL}(\Sigma)$ formulas $\psi_{R, p, q}(X, x, y)$ so that for every graph G : $G \models \psi_{R, p, q}(X, x, y)$ if and only if there exists a run of R on G starting from configuration $\langle x, p \rangle$ that has a prefix that reaches the configuration $\langle y, q \rangle$ and the set of positions on the prefix is exactly X .

We can build $\text{MSOL}(\Sigma)$ formulas $\psi_{R, p, q}^\infty(X, x, y)$ so that for every graph G : $G \models \psi_{R, p, q}^\infty(X, x, y)$ if and only if there is a run of R on G starting from configuration $\langle x, p \rangle$ that reaches the configuration $\langle y, q \rangle$ infinitely often and the set of positions on the run is exactly X .

⁸Note that in our approach $\phi_{\bar{R}, T}$ does not depend on \mathcal{R} or on \mathcal{G} .

PROOF. A robot $R = (Q, \delta)$ is a finite automaton (without initial or final states) over a finite alphabet Alph of instructions, i.e., $\text{Alph} \subset \{\uparrow_\sigma : \sigma \in \Sigma\} \cup \text{MSOL}_1(\Sigma)$. By Kleene’s theorem — which states that every finite-state automaton can be translated into a regular expression — we can build a regular expression exp (that depends on R, p, q) over alphabet Alph for the language of the automaton R with initial state p and final state q .

By induction on the structure of regular expressions over alphabet Alph we build MSOL formulas:

- $\varphi_\emptyset := \text{false}$
- $\varphi_\epsilon(X, x, y) := x = y \wedge x \in X$
- $\varphi_{\uparrow_\sigma}(X, x, y) := \text{edg}_\sigma(x, y) \wedge x, y \in X$
- $\varphi_\tau(X, x, y) := x = y \wedge \tau(x) \wedge x \in X$
- $\varphi_{r+s} := \varphi_r \vee \varphi_s$
- $\varphi_{r \cdot s}(X, x, y) := \exists z [\varphi_r(X, x, z) \wedge \varphi_s(X, z, y)]$
- $\varphi_{r^*}(X, x, y) := \forall Z[(\text{cl}_{\phi_r}(X, Z) \wedge x \in Z) \rightarrow y \in Z]$
where
- $\text{cl}_\phi(X, Z) := \forall a, b [(a \in Z \wedge \phi(X, a, b)) \rightarrow b \in Z]$.

An easy induction shows that $G \models \varphi_r(X, x, y)$ if and only if there is a sequence of instructions $\text{ins} \in \text{Alph}^*$ accepted by the regular expression r and a path from x to y that follows instructions ins , and that only visits vertices in X (but not necessarily all of X). Finally, define the MSOL formula $\psi_{R, p, q}(X, x, y)$ to state that $\phi_{\text{exp}}(X, x, y)$ holds and X is minimal: $\phi_{\text{exp}}(X, x, y) \wedge \neg \exists Y (\phi_{\text{exp}}(Y, x, y) \wedge Y \subset X)$.

Similarly, by a variation of Kleene’s Theorem, we can build an ω -regular expression exp over alphabet Alph for the language consisting of all infinite sequences that label infinite paths in R that start in p and see q infinitely often. Then, inductively build φ_{exp} , as before, with the following additional rule:

$$\varphi_{r^\omega}(X, x, y) := \exists z [\varphi_{r^*}(X, x, y) \wedge \varphi_r(X, y, z) \wedge \varphi_{r^*}(X, z, y)]$$

As before, define the MSOL formula $\psi_{R, p, q}^\infty(X, x, y)$ to state that X is minimal such that $\phi_{\text{exp}}(X, x, y)$ holds. \square

The idea of the proof of Lemma 1 follows [7] who built a formula expressing that there is a run starting in configuration $\langle x, p \rangle$ that reaches configuration $\langle y, q \rangle$. Our Lemma extends this in two ways, i.e., recording the visited states X , and expressing that a configuration occurs infinitely often.

NOTE 1. *The case of multiple robots ($k > 1$) is more subtle. For instance, the analogue of Lemma 1 does not hold. Indeed, for $k = 2$, let $R_1 = R_2$ be robots that have one state p and that non-deterministically move in every direction. Then the statement “there is a run of the robots from configuration $\langle x_1, x_2, p, p \rangle$ to configuration $\langle y_1, y_2, p, p \rangle$ ” is equivalent to the statement that the k -ary transitive closure of the edge relation in graph G contains the tuple $\langle x_1, x_2, y_1, y_2 \rangle$. However, the k -ary transitive closure is not expressible in MSOL (as was pointed out in Example MF5 in Section 2).*

4.3 Robot Task Logic — RTL

We now define a logic, called **RTL**, for expressing robot tasks in the case of one robot ($k = 1$).

Syntax. Formulas of **RTL** are built, as in the definition of **MSOL** from Section 2, from the following atomic formulas: $x = y$, $\text{Reach}(X, x, y)$, $\text{Halt}(X, x, y)$, $\text{Infty}(X, x, y)$, and $\text{Rept}(X, x, y)$.

Semantics. Formulas of **RTL** are interpreted with respect to graphs and robots. Thus, given a graph G and a robot R with state set Q , initial-state set I , repeating-state set A , and halting-state set H , define the satisfaction relation \models_{RTL} :

$$\begin{aligned} \langle G, R \rangle \models_{\text{RTL}} \text{Reach}(X, x, y) &\quad \text{iff } G \models \bigwedge_{p \in I} \bigvee_{q \in Q} \psi_{R,p,q}(X, x, y) \\ \langle G, R \rangle \models_{\text{RTL}} \text{Halt}(X, x, y) &\quad \text{iff } G \models \bigwedge_{p \in I} \bigvee_{q \in H} \psi_{R,p,q}(X, x, y) \\ \langle G, R \rangle \models_{\text{RTL}} \text{Infty}(X, x, y) &\quad \text{iff } G \models \bigwedge_{p \in I} \bigvee_{q \in Q} \psi_{R,p,q}^\infty(X, x, y) \\ \langle G, R \rangle \models_{\text{RTL}} \text{Rept}(X, x, y) &\quad \text{iff } G \models \bigwedge_{p \in I} \bigvee_{q \in A} \psi_{R,p,q}^\infty(X, x, y) \end{aligned}$$

The formulas $\psi_{R,p,q}$ and $\psi_{R,p,q}^\infty$ are from Lemma 1. Extend the satisfaction relation to all formulas of **RTL** in the natural way.

Examples. Here are some example **RTL** formulas and their meanings.

1. The atomic formula $\text{Reach}(X, x, y)$ expresses that the robot, starting in position x , reaches position y , and the set of visited vertices is X . The **RTL** formula $\forall x \exists y \text{Reach}(V, x, y)$ expresses that the robot explores the graph, no matter where it starts.
2. The **RTL** formula $\forall x \exists y \text{Halt}(V, x, y)$ expresses “explore and stop”.
3. The **RTL** formula $\forall x \text{Halt}(V, x, x)$ expresses “explore and return”.
4. The atomic formula $\text{Infty}(X, x, y)$ expresses that the robot, starting in position x , visits position y infinitely often, and the set of vertices the robot visits along this run is exactly X . Thus $\forall x \text{Infty}(V, x, x)$ is an **RTL** formula expressing that the robot “perpetually explores” the graph.

A task T is *captured* by an **RTL** formula ϑ if $\langle G, R \rangle \models_{\text{RTL}} \vartheta$ is equivalent to the statement that every run of R on G is in $T(G)$. The example formulas show that the first three tasks from Section 3.1 are captured by **RTL** formulas.

Here is the main theorem that solves the parameterised verification problem in the case of a single robot ($k = 1$). It reduces the **PVP** to **MSOL**-validity of the set of environments.

THEOREM 2. Fix an edge-label set Σ . Let \mathcal{R} be the set of all robots over $\text{INS}_{\Sigma,1}$, let T be a task that is captured by an **RTL** formula, and let \mathcal{G} be a set of Σ -graphs with decidable **MSOL**-validity problem. Then $\text{PVP}_T(\mathcal{G}, \mathcal{R})$ is decidable.

PROOF. Say **RTL** formula ϑ captures T . Given $R \in \mathcal{R}$, build the formula $\phi_{R,T}$ by replacing every atomic formula in

ϑ by its definition with respect to R , e.g., $\text{Reach}(X, x, y)$ is replaced by $\bigwedge_{p \in I} \bigvee_{q \in Q} \psi_{R,p,q}(X, x, y)$. A routine induction on the structure of the formula ϑ gives: $\langle G, R \rangle \models_{\text{RTL}} \vartheta$ if and only if $G \models \phi_{R,T}$. But by Lemma 1 $\phi_{R,T}$ is a formula in **MSOL**(Σ). Thus we can apply the fact that the **MSOL**-validity problem for \mathcal{G} is decidable to decide whether or not $G \models \phi_{R,T}$ for all $G \in \mathcal{G}$. \square

COROLLARY 1. If \mathcal{R} and T are as in Theorem 2 and \mathcal{G} is a context-free set of graphs, then $\text{PVP}_T(\mathcal{G}, \mathcal{R})$ is decidable.

NOTE 2. The case of multiple robots ($k > 1$) is harder to analyse. Indeed, Theorem 1 states that **PVP** is undecidable already for $k = 2$ on lines (which is a very basic context-free set of graphs) and simple reachability tasks. It is a challenging problem to find natural and useful restrictions on robots which allow the analogue of Corollary 1 (or Lemma 1) to hold in the case of multiple robots.

5. ILLUSTRATION: DISTRIBUTED COMPUTING

We now instantiate our framework to a popular model of autonomous mobile-agent from the distributed computing literature, i.e., [27, 21, 16, 33, 28, 18]. To distinguish their agents from ours, we call theirs *DC-robots*. Here is their model: environments are modeled as undirected graphs, and each vertex is annotated with a local port numbering i.e., for every vertex v the set of labels of the edges of v are in bijection with $\{1, 2, \dots, \deg(v)\}$. A DC-robot finding itself at vertex v can decide to exit via local port-number d and update its local state based on a) its current state, b) the identification of the robots at the same vertex v , c) the degree of v , and d) the local-port number of v of the edge it used when arriving at v . Thus graphs are assumed to have bounded degree Δ . Typical tasks from this literature are “perpetual exploration”, “exploration with return”, and “gathering”. The main twist is that the robots should perform their task on a graph no matter the local port numbering.

Such robot systems are easily expressible in our framework. The edge-label set Σ is defined to be $[\Delta]$, the edge-relation E is assumed to be symmetric, and $\lambda(v, w) = i$ codes that i is v 's port number for the undirected edge $\{v, w\}$. Note that $\lambda(v, w)$ need not equal $\lambda(w, v)$. The interesting aspect is how to simulate d) above. Our robot must store in its state both the state of the DC-robot, as well as the local-port number with which it entered the current vertex. It does this as follows: when our robot is at vertex v , and the DC-robot says to take exit i , our robot first determines the w such that $\lambda(v, w) = i$, and then it determines the $j \in \Delta$ such that $\lambda(w, v) = j$. It can do this with test-instructions in **FOL**(Σ).

Rotor-Robot We now give a simple but important example that can be analysed in our framework. The *rotor robot* operates as follows: when it enters a vertex v by port i it leaves by port $i + 1$ modulo $\deg(v)$. This robot is known by various other names: Abelian mobile robot, rotor walk, ant walk, Eulerian walker, and Propp machine. It is important because it is a viable alternative to probabilistic robots that perform random walks [11]. It has the property that it explores all trees — a result that seems to be folklore, and that could be proved, for fixed degree Δ , using Theorem 2.

Label-Guided Tasks. The graphs in this paper are edge-labeled, but our results hold allowing vertex-labels. Moreover, our framework can incorporate questions of the form: given a robot and a task, decide if for every graph $G \in \mathcal{G}$ there exists a labeling of G such that the robot achieves the task on the graph with the help of the labeling. For instance, although there is no DC-robot that perpetually explores all graphs, there is a DC-robot and an algorithm that colours vertices by three colours, such that the robot can perpetually explore every such coloured graph [16]. Since the set of all graphs does not have decidable-validity, we might only hope to verify variations of this fact for restricted classes of graphs, e.g., fix a context-free set of graphs \mathcal{G} ; then one can decide, given a DC-robot, whether or not for every $G \in \mathcal{G}$ the robot achieves the task “there is a colouring of G using three colours that the robot can use to perpetually explore G ”. The reason this fits into Theorem 2 is that this task is captured by an RTL formula — indeed, the property that $X_1, X_2, X_3 \subseteq V$ colour a graph is easily expressed in MSOL (just say that $\wedge_{i \neq j} X_i \cap X_j = \emptyset$ and $X_1 \cup X_2 \cup X_3 = V$).

6. COMPLEXITY CONSIDERATIONS

As discussed in Section 4.2, the framework reduces the parameterised verification problem $\text{PVP}_T(\mathcal{G}, \mathcal{R})$ to the $\Phi_{\bar{R}, T}$ -validity problem for \mathcal{G} where $\Phi_{\bar{R}, T}$ is the set of formulas $\{\phi_{\bar{R}, T} : \bar{R} \in \mathcal{R}\}$. Unfortunately, the complexity of the decision procedure for $\text{PVP}_T(\mathcal{G}, \mathcal{R})$ in Corollary 1 may be non-elementary, i.e., not bounded by any tower of exponentials in the size of the input robot R . Indeed: although the size of the computed formula $\phi_{R, T}$ is exponential in the size of the robot R and linear in the size of the RTL formula capturing T , but the complexity of the MSOL-validity problem for \mathcal{G} is non-elementary even taking \mathcal{G} to be the set of binary-labeled lines [42].

Consequently, we illustrate that improved decision procedures can be found for interesting tasks and sets of graphs.

A robot is *deterministic* if for all $p \in Q$, either a) there exists $q, q' \in Q$ and $\tau \in \text{MSOL}(\Sigma)$ and the only transitions out of p are (p, τ, q) and $(p, \neg\tau, q')$, or b) there exists $\sigma \in \Sigma, q, q' \in Q$ and the only transitions out of p are (p, \uparrow_σ, q) and $(p, \neg\exists z(\text{edg}_\sigma(x, z)), q')$. In other words, a) says that if test τ holds goto state q else goto state q' , and b) says that if there is an edge in the graph labeled σ then traverse it (in case of many such edges, pick one nondeterministically) and go to state q , otherwise goto state q' . A Σ -graph is *deterministic* if $\lambda(v, w) = \lambda(v, w')$ implies $w = w'$. For instance, Δ -ary trees are deterministic. Note that deterministic robots have at most one run on deterministic graphs.

THEOREM 3. Fix $\Delta \in \mathbb{N}$. Let \mathcal{G} be the Δ -ary trees, let \mathcal{R} be all deterministic robots, and let T be the “explore and halt” task. Then $\text{PVP}_T(\mathcal{G}, \mathcal{R})$ is EXPTIME-COMPLETE.

PROOF SKETCH. The idea is to inter-reduce the parameterised verification problem with the universality problem for deterministic tree-walking automata (DTWA). A DTWA is a deterministic machine that can recognise sets of trees: the automaton starts at the root, at any given time the automaton sits on a node of the input tree, can test if the current node is a leaf, the root, or the i th child (for $i \leq \Delta$), and based on these tests the machine updates its internal state and executes one of the commands: “accept the tree”, “reject the tree”, “go to the parent” or “go the i th child”. The universality of DTWA is EXPTIME-COMPLETE. Indeed, from

a DTWA A build a DTWA B that simulates A and rejects whenever A runs forever (this causes a quadratic blowup in the number of states [36]); then complement B by swapping accept and reject states to get a DTWA C ; then convert C into an ordinary frontier-to-root tree automaton D using a subset construction that calculates loops of C [9, Fact 1] (this causes an exponential blowup); then test D for emptiness (which can be done in P).

Here is the reduction that gives the upper bound: given a deterministic robot R build a DTWA A_R that operates on trees t with marked nodes s and v , written (t, s, v) , as follows: first it does a DFS from the root, and when it reaches s it begins the simulation of R ; after the simulation begins, A_R remembers if v is visited; if R enters a halt state and v was visited then A_R accepts (t, s, v) , and if R enters a halt state and v was not visited then A_R rejects input (t, s, v) . Thus: R “explores and halts” iff for all (t, s, v) the run of R on t starting at s visits v and later enters a halt state iff A_R accepts all inputs of the form (t, s, v) . The size of A_R is linear in the size of R .

Here is the reduction that gives the lower bound: given a DTWA A , build a robot R_A which first explores the input tree t (doing, say, a DFS), then simulates A from the root, and halts iff A accepts (thus R_A runs forever if A rejects). Since R_A always explores its input, A accepts t iff R_A explores and halts on t . The size of R_A is linear in the size of A . \square

NOTE 3. For every $\Delta \geq 2$, there is no robot that “explores and halts” on all local port-numbered Δ -ary trees [21]. The proof above is easily adapted to yield an algorithm that, given a robot R and $\Delta \geq 2$, returns a local-port numbered tree on which the robot does not succeed to “explore and halt”.

7. COMPARISON WITH RELATED WORK

Robot systems are considered distributed if they involve autonomous agents with no central control. In light of this, we first describe the relationship of our work to formal verification of distributed systems generally, and then to formal verification of robot protocols in particular.

Formal verification of distributed systems.

Typical parameters that arise in the study of distributed systems are the number of agents, the number of assumed faulty agents, etc. Since parameterised problems of distributed systems are, in general, undecidable [3, 43], the formal methods community developed sound but incomplete methods that require human intervention, e.g., counter- and predicate-abstractions, inductive invariants, regular model checking and acceleration techniques. See for instance [39, references on pages 2-3].

On the other hand, by simplifying the systems one can get decidable PVP. These use techniques from games, automata theory and logic, notably finite-model properties/cutoffs, reduction to Petri nets, and the theory of well-structured transition systems [24, 26, 23, 15, 31, 19, 1, 2].

Of all these models, token-passing systems (i.e., the models in [24, 15, 1, 2]) are the closest to robots — both tokens and robots move along the vertices of graphs. However, we now argue that the model in these cited papers is incomparable with our model. First, the translation of their token-passing systems into robot systems would require that robots can read and write to variables at the vertices (this

is to model the fact that processes have states). However, we assumed environments are static (because otherwise the PVP is quickly undecidable). Conversely, translating robot-systems into the cited token-passing systems requires that the robot-systems satisfy the following unrealistic assumptions (that are used in their decidability proofs): when a robot decides to move, an adversary decides which edge it takes, as well as to which of its internal states to transition (i.e., even the robot’s memory may be scrambled). Not even the simplest robots from the distributed computing literature (e.g., the rotor robot, or the DFS robot) satisfy this double restriction.

Formal verification of robot systems.

The formal methods community has only recently [30, 20, 12, 6, 4, 35] begun explicitly verifying and synthesising robot protocols (rather than distributed systems in general). However, half of these papers only treat small values of the parameters. For instance, one such paper concludes [6, page 11]:

While our method is parameterised by both k [the number of robots] and n [the size of the graph], it does not permit to verify whether a [robot] protocol is valid for every k and n satisfying a particular predicate.

The papers that do treat parameterised robot protocols do not give sound and complete decision procedures for their systems, as we do. Indeed, [4] uses a proof assistant to provide certificates (formal proofs) of impossibility results about robot networks; [35] uses the theory of games on graphs to synthesise a robot protocol for gathering k robots on a ring of size n (for small values of k, n), and relies on a hand-proven induction to prove that the synthesised robot protocol works for all values of the parameters k, n ; and [32] presents a sound technique that may identify cutoffs using a counter-abstraction in order to draw conclusions about certain swarm algorithms, independently of the number of swarming agents.

The quotation above continues:

Adapting recent advances in parameterised model checking [citation elided] would be a nice way to obtain such results.

Both the methodology (reducing parameterised verification of robot systems to validity problems in logic) and the results of this paper (algorithms for automatic parameterised verification of robot systems consisting of a single robot in an unknown environment) are novel and have succeeded where other methods and “recent advances” (discussed in the previous subsection) have not.

Comparison with graph-walking automata.

Graph-walking automata. There is no canonical definition of automaton on graphs. Our model of a single robot is equivalent to graph-walking automata with MSOL-tests [7]. The proof idea of Lemma 1 (which compiles robots into formulas over graphs) is borrowed from [7, 25]. Other classical notions of automata on graphs (e.g., [8]) are often too expressive and lead to undecidable parameterised verification problems (see the proof of Theorem 1).

Multi-head automata. If agents are modeled as finite-state machines, then multi-agent systems are instances of

multi-head automata, and the parameterised verification problem for reachability tasks is equivalent to the universality problem of such automata. A technical difference between our robot-ensembles and multi-head automata is that the k -many heads of a multi-head automaton are operated from a central control. In the language of robots this would mean that the robots can communicate their current local states to each other. Such communication is disallowed as it quickly results in undecidability. However, the proof of Theorem 1 shows that, similarly, even simple vision-based communication leads to undecidability.

Tree-walking automata (TWA) are a natural generalisation to trees of two-way automata on words. Tree-walking automata with a single head, and their corresponding regular expressions, have been studied for their own sake [9], in formal verification, e.g., [46, 10, 37], and as tree pattern-matching tools [13, 41]. TWA are used in Theorem 3 to reason about a single robot walking on unknown trees.

8. FUTURE CHALLENGES

This paper takes a step in the following general research agenda: find natural robot systems that have decidable or tractable PVP.

Regarding decidability, much work remains to be done. An important problem is to extend the methodology or results of this paper to multiple robots. Notes 1 and 2 discuss the challenges facing such an investigation.

Similarly, in light of the fact that PVP is quickly undecidable in a dynamic environment (e.g., this is the case for simple reachability tasks of a single robot that can read and write to every vertex on a line, cf. [43, 24]), what restrictions on the robot or the dynamic environment will result in decidable PVP?

On the other hand, we believe it is feasible to extend our framework to quantitative tasks, such as minimising the number of steps to complete a task.

Regarding complexity, we have seen that our general algorithm for solving the PVP (in Theorem 2) has high computational complexity, and we have seen (in Theorem 3) that a certain problem on trees is EXPTIME-COMPLETE. For which systems (tasks, classes of graphs, and classes of robots) is the PVP solvable in P, NP, or PSPACE?

We believe that tackling these questions will open avenues both in automata theory and in the verification of mobile multi-agent systems in unknown environments.

Acknowledgements

I thank the reviewers for their careful reading and provocative questions which helped improve the content and presentation of this paper.

REFERENCES

- [1] B. Aminof, S. Jacobs, A. Khalimov, and S. Rubin. Parameterized model checking of token-passing systems. In *VMCAI*, pages 262–281, 2014.
- [2] B. Aminof, T. Kotek, F. Spegni, S. Rubin, and H. Veith. Parameterized model checking of rendezvous systems. In *CONCUR*, pages 109–124, 2014.
- [3] K. Apt and D. Kozen. Limits for automatic verification of finite-state concurrent systems. *Inf. Process. Lett.*, 15:307–309, 1986.

- [4] C. Auger, Z. Bouzid, P. Courtieu, S. Tixeuil, and X. Urbain. Certified impossibility results for byzantine-tolerant mobile robots. In *SSS*, pages 178–190, 2013.
- [5] M. A. Bender, A. Fernandez, D. Ron, A. Sahai, and S. Vadhan. The power of a pebble: Exploring and mapping directed graphs. *Information and Computation*, 176(1):1–21, 2002.
- [6] B. Berard, L. Millet, M. Potop-Butucaru, Y. Thierry-Mieg, and S. Tixeuil. Formal verification of mobile robot protocols. Report <hal-00834061>, 2013.
- [7] R. Bloem and J. Engelfriet. Monadic second order logic and node relations on graphs and trees. In *Structures in Logic and Computer Science*, pages 144–161, 1997.
- [8] M. Blum and C. Hewitt. Automata on a 2-dimensional tape. *SWAT (FOCS)*, pages 155–160, 1967.
- [9] M. Bojańczyk. Language and automata theory and applications. *Lecture Notes in Computer Science Volume 5196*, pages 1–2, 2008.
- [10] P. A. Bonatti, C. Lutz, A. Murano, and M. Y. Vardi. The complexity of enriched μ -calculi. *Logical Methods in Computer Science*, 4(3:11):1–27, 2008.
- [11] B. Bond and L. Levine. Abelian networks: foundations and examples. *CoRR*, abs/1309.3445, 2013.
- [12] F. Bonnet, X. Defago, F. Petit, M. Potop-Butucaru, and S. Tixeuil. Brief announcement: Discovering and assessing fine-grained metrics in robot networks protocols. In *SSS*, pages 282–284. 2012.
- [13] A. Brüggemann-Klein and D. Wood. Caterpillars: A context specification technique. *Markup Languages*, 2:81–106, 2000.
- [14] E. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 1999.
- [15] E. Clarke, M. Talupur, T. Touili, and H. Veith. Verification by network decomposition. In *CONCUR 2004*, volume 3170, pages 276–291, 2004.
- [16] R. Cohen, P. Fraigniaud, D. Ilcinkas, A. Korman, and D. Peleg. Label-guided graph exploration by a finite automaton. In *ICALP*, pages 335–346. 2005.
- [17] B. Courcelle and J. Engelfriet. Book: Graph structure and monadic second-order logic. a language-theoretic approach. *Bull. EATCS*, 108:179, 2012.
- [18] S. Das. Mobile agents in distributed computing: Network exploration. *Bull. EATCS*, 109:54–69, 2013.
- [19] G. Delzanno. Parameterized verification and model checking for distributed broadcast protocols. In *ICGT*, pages 1–16, 2014.
- [20] S. Devismes, A. Lamani, F. Petit, P. Raymond, and S. Tixeuil. Optimal grid exploration by asynchronous oblivious robots. *CoRR*, abs/1105.2461, 2011.
- [21] K. Diks, P. Fraigniaud, E. Kranakis, and A. Pelc. Tree exploration with little memory. *Journal of Algorithms*, 51(1):38–63, 2004.
- [22] H. Ebbinghaus and J. Flum. *Finite model theory*. Perspectives in Mathematical Logic. Springer, 1995.
- [23] E. Emerson and V. Kahlon. Model checking guarded protocols. In *LICS*, pages 361–370, 2003.
- [24] E. Emerson and K. Namjoshi. On reasoning about rings. *Int. J. Found. Comput. Sci.*, 14(4):527–550, 2003.
- [25] J. Engelfriet and H. J. Hoogeboom. Nested pebbles and transitive closure. In *STACS*, pages 477–488, 2006.
- [26] J. Esparza, A. Finkel, and R. Mayr. On the verification of broadcast protocols. In *LICS*, pages 352–359, 1999.
- [27] P. Fraigniaud, D. Ilcinkas, G. Peer, A. Pelc, and D. Peleg. Graph exploration by a finite automaton. *Theoretical Computer Science*, 345:331 – 344, 2005.
- [28] L. Gasieniec and T. Radzik. Memory efficient anonymous graph exploration. In H. Broersma, T. Erlebach, T. Friedetzky, and D. Paulusma, editors, *Graph-Theoretic Concepts in Computer Science*, volume 5344 of *LNCS*, pages 14–29. 2008.
- [29] E. Grädel, W. Thomas, and T. Wilke, editors. *Automata, Logics, and Infinite Games: A Guide to Current Research*, volume 2500 of *LNCS*. Springer, 2002.
- [30] X. Huang, P. Maupin, and R. Van Der Meyden. Model checking knowledge in pursuit evasion games. In *IJCAI*, pages 240–245, 2011.
- [31] P. Kouvaros and A. Lomuscio. Automatic verification of parameterised multi-agent systems. In *AAMAS*, pages 861–868, 2013.
- [32] P. Kouvaros and A. Lomuscio. A counter abstraction technique for the verification of robot swarms. To appear in *AAAI*, 2015.
- [33] E. Kranakis, D. Krizanc, and S. Rajsbaum. Mobile agent rendezvous: A survey. *SIROCCO*, pages 1–9, 2006.
- [34] E. Kranakis, D. Krizanc, and S. Rajsbaum. Computing with mobile agents in distributed networks. In S. Rajasekaran and J. Reif, editors, *Handbook of Parallel Computing: Models, Algorithms, and Applications*, pages 8–1 to 8–20. Chapman Hall/CRC Computer and Inf. Sci. Series, 2007.
- [35] L. Millet, M. Potop-Butucaru, N. Sznajder, and S. Tixeuil. On the synthesis of mobile robots algorithms: The case of ring gathering. In *SSS*, pages 237–251, 2014.
- [36] A. Muscholl, M. Samuelides, and L. Segoufin. Complementing deterministic tree-walking automata. *Inf. Process. Lett.*, 99(1):33–39, 2006.
- [37] J. Obdržálek. Fast mu-calculus model checking when tree-width is bounded. In *CAV*, pages 80–92, 2003.
- [38] A. Pelc. Disc 2011 invited lecture: Deterministic rendezvous in networks: Survey of models and results. In *DISC*, pages 1–15, 2011.
- [39] A. Pnueli, J. Xu, and L. Zuck. Liveness with $(0,1,\infty)$ -counter abstraction. In *CAV*, pages 93–111, 2002.
- [40] A. Rosenberg. On multi-head finite automata. *IBM Journal of Research and Development*, 10(5):388–394, Sep 1966.
- [41] T. Schwentick. Foundations of xml based on logic and automata: A snapshot. In *FoIKS*, pages 23–33, 2012.
- [42] L. J. Stockmeyer. *The complexity of decision problems in automata theory and logic*. PhD thesis, MIT, 1974.
- [43] I. Suzuki. Proving properties of a ring of finite-state machines. *Inf. Process. Lett.*, 28(4):213–214, July 1988.
- [44] W. Thomas. Automata on infinite objects. In *Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics*, pages 133–192, 1990.

- [45] W. Thomas. Languages, automata, and logic. In *Handbook of Formal Languages*, pages 389–455, 1996.
- [46] M. Y. Vardi. Reasoning about the past with two-way automata. In *ICALP*, pages 628–641, 1998.

Graded Strategy Logic: Reasoning about Uniqueness of Nash Equilibria *

Benjamin Aminof

Vadim Malvone, Aniello Murano, Sasha Rubin

Technische Universität Wien, Austria

Università degli Studi di Napoli Federico II, Italy

benj@forsyte.tuwien.ac.at

{malvone,murano,rubin}@unina.it

Strategy Logic (SL) is a well established formalism for strategic reasoning in multi-agent systems. It is built over LTL and treats strategies as first-order objects that are associated with agents by means of a binding operator. In this work, we introduce Graded Strategy Logic (GRADED-SL), an extension of SL by graded quantifiers over tuples of strategy variables such as “there exist at least g different tuples (x_1, \dots, x_n) of strategies”. We study the model-checking problem of GRADED-SL and prove that it is no harder than for SL, i.e., it is non-elementary in the quantifier rank. We show that GRADED-SL allows one to count the number of different strategy profiles that are Nash equilibria (NE), or subgame-perfect equilibria (SPE). By analyzing the structure of the specific formulas involved, we conclude that the important problems of checking for the existence of a unique NE or SPE can both be solved in 2EXPTIME, which is not harder than merely checking for the existence of such equilibria.

1 Introduction

Strategy Logic (SL) is a powerful formalism for reasoning about strategies in multi-agent systems [24, 25]. Strategies tell an agent what to do — they are functions that prescribe an action based on the history. The key idea in SL is to treat strategies as first-order object. A strategy x can be quantified existentially $\langle\langle x \rangle\rangle$ (read: there exists a strategy x) and universally $\llbracket x \rrbracket$ (read: for all strategies x). Strategies are not intrinsically glued to specific agents: the *binding* operator (α, x) allows one to bind an agent α to the strategy x . SL strictly subsumes several other logics for strategic reasoning including the well known ATL and ATL* [2]. Being a very powerful logic, SL can directly express many solution concepts [11, 17, 18, 24] among which that a strategy profile \bar{x} is a Nash equilibrium, and thus also the existence of a Nash equilibrium (NE) as well as other important solution concepts such as subgame-perfect equilibrium (SPE) [32].

Nash equilibrium is one of the most important concepts in game theory, forming the basis of much of the recent fundamental work in multi-agent decision making. A challenging and important aspect is to establish whether a game admits a *unique* NE [1]. This problem impacts on the predictive power of NE since, in case there are multiple equilibria, the outcome of the game cannot be uniquely pinned down [12]. Unfortunately, uniqueness has mainly been established either for special cost functions [1], or for very restrictive game topologies [27]. More specifically, uniqueness of NE in game theory is proved with procedures that are separated from the ones that check for its existence [1]; these procedures require various transformations of the best response functions of individual contributors [14, 15], and there is no general theory that can be applied to different application areas [1].

In this paper we address and solve the problem of expressing the uniqueness of certain solution concepts (and NE in particular) in a principled and elegant way. We introduce an extension of SL called

*This is an expository paper reporting on the work [4].

GRADED-SL and study its model-checking problem. We extend SL by replacing the quantification $\langle\langle x \rangle\rangle$ and $[[x]]$ over strategy variables with *graded quantification over tuples of strategy variables*: $\langle\langle x_1, \dots, x_n \rangle\rangle^{\geq g}$ (read $\langle\langle x_1, \dots, x_n \rangle\rangle^{\geq g}$ as “there exist at least g different tuples (x_1, \dots, x_n) of strategies”) and its dual $[[x_1, \dots, x_n]]^{< g}$ ($g \in \mathbb{N}$). Here, two strategies are different if they disagree on some history. That is, we count strategies syntactically as is usual in graded extensions of modal and description logics [8, 9, 16, 19]. The key for expressing uniqueness of NE is the combination of quantifying over tuples (instead of singleton variables), and adding counting (in the form of graded modalities).

We address the model-checking problem for GRADED-SL and prove that it has the same complexity as SL, i.e., it is non-elementary in the nesting depth of quantifiers. In particular, we show that model checking GRADED-SL formulas with a nesting depth $k > 0$ of blocks of quantifiers (a block of quantifiers is a maximally-consecutive sequence of quantifiers of the same type, i.e., either all existential, or all universal) is in $(k+1)\text{EXPTIME}$, and that for the special case where the formula starts with a block of quantifiers, it is in $k\text{EXPTIME}$. Since many natural formulas have a small number of quantifiers, and even smaller nesting depth of blocks of quantifiers, the complexity of the model-checking problem is not as bad as it seems. Specifically, several solution concepts can be expressed as SL formulas with a small number of quantifiers [11, 17, 18, 24]. Since the existence of a NE, and the fact that there is at most one NE, can be expressed in GRADED-SL using simple formulas we are able to conclude that the problem of checking the uniqueness of a NE can be solved in 2EXPTIME . Previously, it was only known that existence of NE can be checked in 2EXPTIME [17, 24], and indeed it is 2EXPTIME -complete [3, 17, 28]. Thus, GRADED-SL is the first logic that can solve the existence and uniqueness of NE (as well as many other solution concepts) in a uniform way.

SL has a few natural and powerful syntactic fragments. In particular, the Nested-Goal fragment of SL requires that bindings and quantifications appear in exhaustive blocks (see Section 2 for details), and can express NE [24]. Similarly, we define Nested-Goal GRADED-SL, a syntactic fragment of GRADED-SL. We show that Nested-Goal GRADED-SL has the same model-checking complexity as Nested-Goal SL, i.e., non-elementary in the *alternation number* of the quantifiers appearing in the formula (the alternation number of a Nested-Goal formula is, roughly speaking, the maximum number of existential/universal quantifier switches in a consecutive sequence of quantifiers, see Appendix A). Since uniqueness of subgame-perfect equilibria (SPE) can be expressed in Nested-Goal GRADED-SL using alternation one, we get a proof that checking the SPE uniqueness is in 2EXPTIME (as well as an alternative proof for NE).

We focus on the *iterated prisoner’s dilemma* (IPD) [29]. The prisoner’s dilemma (PD) is a popular metaphor for the problem of stable cooperation and has been widely used in several application domains [5]. In the classic definition, it consists of two players, each of them has an option to defect or collaborate. More involved is the IPD in which the process is repeated and one can model reactive strategies in continuous play. The IPD has become a standard model for the evolution of cooperative behaviour within a community of egoistic agents, frequently cited for implications in both sociology and biology (see [5]). Our work shows that checking the uniqueness of a NE and of a SPE in an IPD can be solved in 2EXPTIME .

The work closest to ours is [22]: also motivated by counting NE, it introduces a graded extension of SL, called GSL. In contrast with our work, GSL has a very intricate way of counting strategies: it gives a semantic definition of equivalence of strategies, and the graded modalities count non-equivalent strategies. While this approach is sound, it heavily complicates the model-checking problem. Indeed, only a very weak fragment of GSL has been solved in [22] by exploiting an *ad hoc* solution that does not seem to be scalable to (all of) GSL.

2 Graded Strategy Logic

In this section we introduce Graded Strategy Logic, which we call GRADED-SL for short.

2.1 Models

Sentences of GRADED-SL are interpreted over *concurrent game structures*, just as for ATL and SL [2, 24].

Definition 2.1 A concurrent game structure (CGS) is a tuple $\mathcal{G} \triangleq \langle AP, Ag, Ac, St, s_I, ap, tr \rangle$, where AP , Ag , Ac , and St are the sets of atomic propositions, agents, actions and states, respectively, $s_I \in St$ is the initial state, and $ap : St \rightarrow 2^{AP}$ is the labeling function mapping each state to the set of atomic propositions true in that state. Let $Dc \triangleq Ag \rightarrow Ac$ be the set of decisions, i.e., functions describing the choice of an action by every agent. Then, $tr : Dc \rightarrow (St \rightarrow St)$ denotes the transition function mapping every decision $\delta \in Dc$ to a function $tr(\delta) : St \rightarrow St$.

We will usually take the set Ag of agents to be $\{\alpha_1, \dots, \alpha_n\}$. A path (from s) is a finite or infinite non-empty sequence of states $s_1 s_2 \dots$ such that $s = s_1$ and for every i there exists a decision δ with $tr(\delta)(s_i) = s_{i+1}$. The set of paths starting with s is denoted $Pth(s)$. The set of finite paths from s , called the histories (from s), is denoted $Hst(s)$. A strategy (from s) is a function $\sigma \in Str(s) \triangleq Hst(s) \rightarrow Ac$ that prescribes which action has to be performed given a certain history. We write Pth , Hst , Str for the set of all paths, histories, and strategies (no matter where they start). We use the standard notion of equality between strategies, [21], i.e., $\sigma_1 = \sigma_2$ iff for all $\rho \in Hst$, $\sigma_1(\rho) = \sigma_2(\rho)$. This extends to equality between two n -tuples of strategies in the natural way, i.e., coordinate-wise.

2.2 Syntax

GRADED-SL extends SL by replacing the singleton strategy quantifiers $\langle\langle x \rangle\rangle$ and $[[x]]$ with the graded (tupled) quantifiers $\langle\langle x_1, \dots, x_n \rangle\rangle^{\geq g}$ and $[[x_1, \dots, x_n]]^{<g}$, respectively, where each x_i belongs to a countable set of variables Vr and $g \in \mathbb{N}$ is called the *degree* of the quantifier. Intuitively, these are read as “there exist at least g tuples of strategies (x_1, \dots, x_n) ” and “all but less than g many tuples of strategies”, respectively. The syntax (α, x) denotes a *binding* of the agent α to the strategy x .

Definition 2.2 GRADED-SL formulas are built inductively by means of the following grammar, where $p \in AP$, $\alpha \in Ag$, $x, x_1, \dots, x_n \in Vr$ such that $x_i \neq x_j$ for $i \neq j$, and $g, n \in \mathbb{N}$:

$$\varphi := p \mid \neg\varphi \mid \varphi \vee \varphi \mid X\varphi \mid \varphi U \varphi \mid \langle\langle x_1, \dots, x_n \rangle\rangle^{\geq g} \varphi \mid (\alpha, x)\varphi.$$

Notation. Whenever we write $\langle\langle x_1, \dots, x_n \rangle\rangle^{\geq g}$ we mean that $x_i \neq x_j$ for $i \neq j$. Shorthands are derived as usual. Specifically, $true \triangleq p \vee \neg p$, $false \triangleq \neg true$, $F\varphi \triangleq true U \varphi$, and $G\varphi \triangleq \neg F \neg \varphi$. Also, we have that $[[x_1, \dots, x_n]]^{<g} \varphi \triangleq \neg \langle\langle x_1, \dots, x_n \rangle\rangle^{\geq g} \neg \varphi$.

In order to define the semantics, we first define the concept of *free placeholders* in a formula, which refer to agents and variables. Intuitively, an agent or variable is free in φ if it does not have a strategy associated with it (either by quantification or binding) but one is required in order to ascertain if φ is true or not. The set of *free agents* and *free variables* of a GRADED-SL formula φ is given by the function $free : GRADED-SL \rightarrow 2^{Ag \cup Vr}$ defined as follows:

- $free(p) \triangleq \emptyset$, where $p \in AP$;
- $free(\neg\varphi) \triangleq free(\varphi)$;
- $free(\varphi_1 \vee \varphi_2) \triangleq free(\varphi_1) \cup free(\varphi_2)$;

- $\text{free}(\mathsf{X} \varphi) \triangleq \text{Ag} \cup \text{free}(\varphi);$
- $\text{free}(\varphi_1 \cup \varphi_2) \triangleq \text{Ag} \cup \text{free}(\varphi_1) \cup \text{free}(\varphi_2);$
- $\text{free}(\langle\langle x_1, \dots, x_n \rangle\rangle^{\geq g} \varphi) \triangleq \text{free}(\varphi) \setminus \{x_1, \dots, x_n\};$
- $\text{free}((\alpha, x)\varphi) \triangleq \text{free}(\varphi), \text{ if } \alpha \notin \text{free}(\varphi), \text{ where } \alpha \in \text{Ag} \text{ and } x \in \text{Vr};$
- $\text{free}((\alpha, x)\varphi) \triangleq (\text{free}(\varphi) \setminus \{\alpha\}) \cup \{x\}, \text{ if } \alpha \in \text{free}(\varphi), \text{ where } \alpha \in \text{Ag} \text{ and } x \in \text{Vr}.$

A formula φ without free agents (resp., variables), i.e., with $\text{free}(\varphi) \cap \text{Ag} = \emptyset$ (resp., $\text{free}(\varphi) \cap \text{Vr} = \emptyset$), is called *agent-closed* (resp., *variable-closed*). If φ is both agent- and variable-closed, it is called a *sentence*.

A formula φ without free agents (resp., variables), i.e., with $\text{free}(\varphi) \cap \text{Ag} = \emptyset$ (resp., $\text{free}(\varphi) \cap \text{Vr} = \emptyset$), is called *agent-closed* (resp., *variable-closed*). Also, φ is a *sentence* if it is both agent- and variable-closed.

SL has a few natural syntactic fragments, the most powerful of which is called Nested-Goal SL. Similarly, we define *Nested-Goal GRADED-SL* (abbreviated GRADED-SL[NG]), as a syntactic fragment of GRADED-SL. As in SL[NG], in GRADED-SL[NG] we require that bindings and quantifications appear in exhaustive blocks. I.e., whenever there is a quantification over a variable in a formula ψ it is part of a consecutive sequence of quantifiers that covers all of the free variables that appear in ψ , and whenever an agent is bound to a strategy then it is part of a consecutive sequence of bindings of all agents to strategies. Finally, formulas with free agents are not allowed. To formalize GRADED-SL[NG] we first introduce some notions. A *quantification prefix* over a finite set $V \subseteq \text{Vr}$ of variables is a sequence $\wp \in \{\langle\langle x_1, \dots, x_n \rangle\rangle^{\geq g}, [[x_1, \dots, x_n]]^{< g} : x_1, \dots, x_n \in V \wedge g \in \mathbb{N}\}^*$ such that each $x \in V$ occurs exactly once in \wp . A *binding prefix* is a sequence $\flat \in \{(a, x) : \alpha \in \text{Ag} \wedge x \in \text{Vr}\}^*$ such that each $\alpha \in \text{Ag}$ occurs exactly once in \flat . We denote the set of binding prefixes by Bn , and the set of quantification prefixes over V by $\text{Qn}(V)$.

Definition 2.3 GRADED-SL[NG] formulas are built inductively using the following grammar, with $p \in \text{AP}$, $\wp \in \text{Qn}(V)$ ($V \subseteq \text{Vr}$), and $\flat \in \text{Bn}$:

$$\varphi ::= p \mid \neg\varphi \mid \varphi \vee \varphi \mid \mathsf{X} \varphi \mid \varphi \cup \varphi \mid \wp\varphi \mid \flat\varphi,$$

where in the rule $\wp\varphi$ we require that φ is agent-closed and $\wp \in \text{Qn}(\text{free}(\varphi))$.

Formulas of GRADED-SL[NG] can be classified according to their *alternation number*, i.e., the maximum number of quantifier switches in a quantification prefix. See Appendix A for the formal definition.

The *quantifier rank* of φ is the maximum nesting of quantifiers in φ , e.g., $\langle\langle x_1, \dots, x_n \rangle\rangle^{\geq g}(\alpha_1, x_1) \dots (\alpha_n, x_n) \wedge_{i=1}^n (\langle\langle y \rangle\rangle(\alpha_i, y)\psi_i) \rightarrow \psi_i$ has quantifier rank 2 if each ψ_i is quantifier free. Moreover, a *quantifier-block* of φ is a maximally-consecutive sequence of quantifiers in φ of the same type (i.e., either all existential, or all universal). The *quantifier-block rank* of φ is exactly like the quantifier rank except that a quantifier block of j quantifiers contributes 1 instead of j to the count.

We conclude this subsection by introducing *One-Goal GRADED-SL*, written GRADED-SL[1G]. The importance of this fragment in SL stems from the fact that it strictly includes ATL* while maintaining the same complexity for both the model checking and the satisfiability problems, i.e. 2EXPTIME-COMPLETE [23, 24]. However, it is commonly believed that Nash Equilibrium cannot be expressed in this fragment. The definition of GRADED-SL[1G] follows.

Definition 2.4 GRADED-SL[1G] formulas are built inductively using the following grammar, with $p \in \text{AP}$, $\wp \in \text{Qn}(V)$ ($V \subseteq \text{Vr}$), and $\flat \in \text{Bn}$:

$$\varphi ::= p \mid \neg\varphi \mid \varphi \vee \varphi \mid \mathsf{X} \varphi \mid \varphi \cup \varphi \mid \wp\varphi,$$

where \wp is a quantification prefix over $\text{free}(\varphi)$.

2.3 Semantics

As for SL, the interpretation of a GRADED-SL formula requires a valuation of its free placeholders. This is done via *assignments* (*from s*), i.e., functions $\chi \in \text{Asg}(s) \triangleq (\text{Vr} \cup \text{Ag}) \rightarrow \text{Str}(s)$ mapping variables/agents to strategies. We denote by $\chi[e \mapsto \sigma]$, with $e \in \text{Vr} \cup \text{Ag}$ and $\sigma \in \text{Str}(s)$, the assignment that differs from χ only in the fact that e maps to σ . Extend this definition to tuples: for $\bar{e} = (e_1, \dots, e_n)$ with $e_i \neq e_j$ for $i \neq j$, define $\chi[\bar{e} \mapsto \bar{\sigma}]$ to be the assignment that differs from χ only in the fact that e_i maps to σ_i (for each i). For an assignment $\chi \in \text{Asg}(s)$ the (χ, s) -play denotes the path $\pi \in \text{Pth}(s)$ such that for all $i \in \mathbb{N}$, it holds that $\pi_{i+1} = \text{tr}(\text{dc})(\pi_i)$, where $\text{dc}(\alpha) \triangleq \chi(\alpha)(\pi_{\leq i})$ for $\alpha \in \text{Ag}$. The function $\text{play} : \text{Asg} \times \text{St} \rightarrow \text{Pth}$, with $\text{dom}(\text{play}) \triangleq \{(\chi, s) : \chi \in \text{Asg}(s)\}$, maps (χ, s) to the (χ, s) -play $\text{play}(\chi, s) \in \text{Pth}(s)$. The notation $\pi_{\leq i}$ (resp. $\pi_{< i}$) denotes the prefix of the sequence π of length i (resp. $i - 1$). Similarly, the notation π_i denotes the i th symbol of π . Thus, $\text{play}(\chi, s)_i$ is the i th state on the play determined by χ from s . For $\chi \in \text{Asg}(s)$ and $i \in \mathbb{N}$, writing $\rho \triangleq \text{play}(\chi, s)_{\leq i}$ (the prefix of the play up to i) and $t \triangleq \text{play}(\chi, s)_i$ (the last state of ρ) define $\chi_i \in \text{Asg}(t)$ to be the assignment from t that maps $e \in \text{Vr} \cup \text{Ag}$ to the strategy that maps $h \in \text{Hst}(t)$ to the action $\chi(e)(\rho_{< i} \cdot h)$.

The semantics of GRADED-SL mimics the one for SL as given in [24]. Given a CGS \mathcal{G} , for all states $s \in \text{St}$ and assignments $\chi \in \text{Asg}(s)$, the relation $\mathcal{G}, \chi, s \models \varphi$ is defined inductively on the structure of φ . In addition to the SL case, we have the existential quantifier $\langle\langle x_1, \dots, x_n \rangle\rangle^{\geq g} \varphi$ interpreted as follows:

$$\begin{aligned} \mathcal{G}, \chi, s \models \langle\langle x_1, \dots, x_n \rangle\rangle^{\geq g} \varphi &\text{ iff there exist } g \text{ many tuples } \bar{\sigma}_1, \dots, \bar{\sigma}_g \text{ of strategies such that:} \\ &\quad \bar{\sigma}_i \neq \bar{\sigma}_j \text{ for } i \neq j, \text{ and} \\ &\quad \mathcal{G}, \chi[\bar{x} \mapsto \bar{\sigma}_i], s \models \varphi \text{ for } 1 \leq i \leq g. \end{aligned}$$

Intuitively, $\langle\langle x_1, \dots, x_n \rangle\rangle^{\geq g} \varphi$ counts the number of distinct tuples of strategies that satisfy φ .

As usual, if χ and χ' agree on $\text{free}(\varphi)$, then $\mathcal{G}, \chi, s \models \varphi$ if and only if $\mathcal{G}, \chi', s \models \varphi$, i.e., the truth of φ does not depend on the values the assignment takes on placeholders that are not free. Thus, for a sentence φ , we write $\mathcal{G} \models \varphi$ to mean that $\mathcal{G}, \chi, s_I \models \varphi$ for some (equivalently, for all) assignments χ , and where s_I is the initial state of \mathcal{G} .

3 Illustrating GRADED-SL: uniqueness of some solution concepts

In game theory, players have objectives that are summarised in a payoff function that maps plays to real numbers. In order to specify such payoffs we follow a formalisation from [18] called *objective* LTL. This will allow us to model the Prisoner’s Dilemma (PD), probably the most famous game in game-theory, as well as an iterated version (IPD). We then discuss appropriate solution concepts, and show how to express these in GRADED-SL.

Let \mathcal{G} be a CGS with n agents. Let $m \in \mathbb{N}$ and fix, for each agent $\alpha_i \in \text{Ag}$, an *objective* tuple $S_i \triangleq \langle f_i, \varphi_i^1, \dots, \varphi_i^m \rangle$, where $f_i : \{0, 1\}^m \rightarrow \mathbb{N}$, and each φ_i^j is an LTL formula over AP. If π is a play, then agent α_i receives payoff $f_i(\bar{h}) \in \mathbb{N}$ where the j ’th bit \bar{h}_j of \bar{h} is 1 if and only if $\pi \models \varphi_i^j$. For instance, f may count the number of formulas that are true.

Prisoner’s Dilemma – One shot and Iterated

For convenience, we describe the Prisoner’s Dilemma (one-shot and iterated) using the same CGS in Figure 1. The difference is in how we define the objectives. For the one-shot version, the objective of agent α_i is $S_i \triangleq \langle f_i, \varphi_i^1, \varphi_i^2, \varphi_i^3, \varphi_i^4 \rangle$ where $\varphi_i^1 \triangleq X\mathbf{S}_i$, $\varphi_i^2 \triangleq X\mathbf{P}_i$, $\varphi_i^3 \triangleq X\mathbf{R}_i$, and $\varphi_i^4 \triangleq X\mathbf{T}_i$ and f_i returns the value of its input vector interpreted as a binary number, e.g., $f_i(0100) = 4$.

In words, we have two agents α_1 and α_2 . Each agent has two actions: cooperate (C) and defect (D), corresponding, respectively, to the options of remaining silent or confessing. For each possible pair

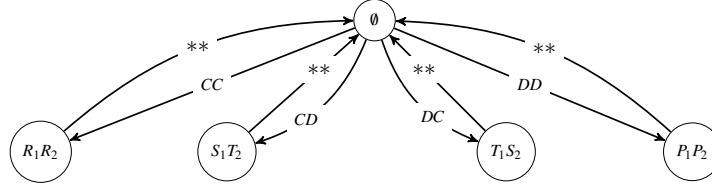


Figure 1: Iterated Prisoner's dilemma.

of moves, the game goes in a state whose atomic propositions represent the payoffs: \mathbf{R}_i represents the *reward* payoff that α_i receives if both cooperate; \mathbf{P}_i is the *punishment* that α_i receives if both defect; \mathbf{T}_i is the *temptation* that α_i receives as a sole defector, and \mathbf{S}_i is the *sucker* payoff that α_i receives as a sole cooperator. The payoffs satisfy the following chain of inequalities: $\mathbf{T}_i > \mathbf{R}_i > \mathbf{P}_i > \mathbf{S}_i$.

The Iterated Prisoner's Dilemma (IPD) is used to model a series of interactions. This is like PD except that after the first choice, both agents have another choice and so on. The main difference between the one-shot and iterated PD is that in the latter the agents' actions (may) depend on the past behaviour. Convenient payoff functions for the IPD are the mean-average and discounted-payoff [7]. Such quantitative payoffs can be replaced, in a first approximation, by LTL payoffs such as “maximise the largest payoff I receive infinitely often”. This is formalised, for agent α_i , by the objective $S_i \triangleq \langle f_i, \varphi_i^1, \varphi_i^2, \varphi_i^3, \varphi_i^4 \rangle$ where $\varphi_i^1 \triangleq GFS_i$, $\varphi_i^2 \triangleq GF\mathbf{P}_i$, $\varphi_i^3 \triangleq GFR_i$, and $\varphi_i^4 \triangleq GFT_i$, and f_i is as in the one-shot PD.

Solution Concepts. Solution concepts are criteria by which one captures what rational agents would do. This is especially relevant in case each agent has its own objective. The central solution concept in game theory is the Nash Equilibrium. A tuple of strategies, one for each player, is called a *strategy profile*. A strategy profile is a *Nash equilibrium (NE)* if no agent can increase his payoff by unilaterally choosing a different strategy. A game may have zero, one, or many NE.

Consider first a CGS \mathcal{G} with n agents, where the objective of the agent $\alpha_i \in Ag$ contains a single LTL formula φ_i (with a larger payoff if it holds than if it doesn't). It is not hard to see that the following formula expresses that $\bar{x} \triangleq (x_1 \dots x_n)$ is a Nash Equilibrium:

$$\phi_{NE}^1(\bar{x}) \triangleq [[y_1]] \dots [[y_n]] \bigwedge_{i=1}^n (\flat_i \varphi_i) \rightarrow \flat \varphi_i$$

where $\flat = (\alpha_1, x_1) \dots (\alpha_n, x_n)$, and $\flat_i = (\alpha_1, x_1) \dots (\alpha_{i-1}, x_{i-1})(\alpha_i, y_i)(\alpha_{i+1}, x_{i+1}) \dots (\alpha_n, x_n)$.

Consider now the general case, where each agent α_i has an objective tuple $S_i \triangleq \langle f_i, \varphi_i^1, \dots, \varphi_i^m \rangle$. Given a vector $\bar{h} \in \{0, 1\}^m$, let $gd_i(\bar{h}) \triangleq \{\bar{t} \in \{0, 1\}^m \mid f_i(\bar{t}) \geq f_i(\bar{h})\}$ be the set of vectors \bar{t} for which the payoff for agent α_i is not worse than for \bar{h} . Also, let $\eta_i^{\bar{h}}$ be the formula obtained by taking a conjunction of the formulas $\varphi_i^1, \dots, \varphi_i^m$ or their negations according to \bar{h} , i.e., by taking φ_i^j if the j 'th bit in \bar{h} is 1, and otherwise taking $\neg \varphi_i^j$. Formally, $\eta_i^{\bar{h}} \triangleq \bigwedge_{j \in \{1 \leq j \leq m \mid h_j = 1\}} \varphi_i^j \wedge \bigwedge_{j \in \{1 \leq j \leq m \mid h_j = 0\}} \neg \varphi_i^j$. Observe that the following formula says that $\bar{x} \triangleq (x_1 \dots x_n)$ is a Nash Equilibrium:

$$\phi_{NE}(\bar{x}) \triangleq [[y_1]] \dots [[y_n]] \bigwedge_{i=1}^n \bigwedge_{\bar{h} \in \{0, 1\}^m} (\flat_i \eta_i^{\bar{h}}) \rightarrow \bigvee_{\bar{t} \in gd_i(\bar{h})} \flat \eta_i^{\bar{t}}$$

Going back to the PD example, due to the simplicity of the payoff functions, we have that:

$$\phi_{PD}(\bar{x}) \triangleq [[y_1]] \dots [[y_n]] \bigwedge_{i=1}^2 \bigwedge_{j=1}^4 (\flat_i \varphi_i^j) \Rightarrow (\bigvee_{r \geq j} \flat \varphi_i^r)$$

As it turns out (again due to the simplicity of the payoff functions), the formula above is also correct for the IPD. It has been argued (in [18, 32]) that NE may be implausible when used for sequential games (of which iterated one shot games are central examples), and that a more robust notion is subgame-perfect equilibrium [30]. Given a game \mathcal{G} , a strategy profile is a *subgame-perfect equilibrium (SPE)* if for every possible history of the game, the strategies are an NE. The following formula expresses that $\bar{x} \triangleq (\alpha_1, x_1) \dots (\alpha_n, x_n)$ is an SPE:

$$\phi_{SPE}(\bar{x}) \triangleq [[z_1, \dots, z_n]](\alpha_1, z_1) \dots (\alpha_n, z_n) G \phi_{NE}(\bar{x})$$

Using graded modalities, we can thus express the uniqueness of a NE using the following GRADED-SL[NG] formula:

$$\langle\langle x_1, \dots, x_n \rangle\rangle^{\geq 1} \phi_{NE}(\bar{x}) \wedge \neg \langle\langle x_1, \dots, x_n \rangle\rangle^{\geq 2} \phi_{NE}(\bar{x})$$

By replacing ϕ_{NE} with ϕ_{SPE} in the formula above, we can express the uniqueness of a SPE.

4 The Model-checking procedure

In this section we study the model-checking problem for GRADED-SL and show that it is decidable with a time-complexity that is non-elementary (i.e., not bounded by any fixed tower of exponentials). However, it is elementary if the number of blocks of quantifiers is fixed. For the algorithmic procedures, we follow an *automata-theoretic approach* [20], reducing the decision problem for the logic to the emptiness problem of an automaton. The procedure we propose here extends that used for SL in [24]. The only case that is different is the new graded quantifier over tuples of strategies.

We start with the central notions of automata theory, and then show how to convert a GRADED-SL sentence φ into an automaton that accepts exactly the (tree encodings) of the concurrent game structures that satisfy φ . This is used to prove the main result about GRADED-SL model checking.

Automata Theory. A Σ -labeled Υ -tree T is a pair $\langle T, V \rangle$ where $T \subseteq \Upsilon^+$ is prefix-closed (i.e., if $t \in T$ and $s \in \Upsilon^+$ is a prefix of t then also $s \in T$), and $V : T \rightarrow \Sigma$ is a labeling function. Note that every word $w \in \Upsilon^+ \cup \Upsilon^\omega$ with the property that every prefix of w is in T , can be thought of as a path in T . Infinite paths are called *branches*. *Nondeterministic tree automata* (NTA) are a generalization to infinite trees of the classical automata on words [31]. *Alternating tree automata* (ATA) are a further generalization of nondeterministic tree automata [13]. Intuitively, on visiting a node of the input tree, while an NTA sends exactly one copy of itself to each of the successors of the node, an ATA can send several copies to the same successor. We use the parity acceptance condition [20]. *Alternating parity tree automata* (APT) are defined in Appendix B. The *language* $L(\mathcal{A})$ of the APT \mathcal{A} is the set of trees T accepted by \mathcal{A} . Two automata are *equivalent* if they have the same language. The *emptiness problem* for alternating parity tree-automata is to decide, given \mathcal{A} , whether $L(\mathcal{A}) = \emptyset$. The *universality problem* is to decide whether \mathcal{A} accepts all trees.

4.1 From Logic to Automata

Following an automata-theoretic approach, we reduce the model-checking problem of GRADED-SL to the emptiness problem for alternating parity tree automata [24]. The main step is to translate every GRADED-SL formula φ (i.e., φ may have free placeholders), concurrent-game structure \mathcal{G} , and state s , into an APT that accepts a tree if and only if the tree encodes an assignment χ such that $\mathcal{G}, \chi, s \models \varphi$.

We first describe the encoding, following [24]. Informally, the CGS \mathcal{G} is encoded by its “tree-unwinding starting from s ” whose nodes represent histories, i.e., the St-labeled St-tree $T \triangleq \langle Hst(s), u \rangle$

such that $u(h)$ is the last symbol of h . Then, every strategy $\chi(e)$ with $e \in \text{free}(\varphi)$ is encoded as an Ac-labelled tree over the unwinding. The unwinding and these strategies $\chi(e)$ are viewed as a single $(\text{VAL} \times \text{St})$ -labeled tree where $\text{VAL} \triangleq \text{free}(\varphi) \rightarrow \text{Ac}$.

Definition 4.1 *The encoding of χ (w.r.t. φ, \mathcal{G}, s) is the $(\text{VAL} \times \text{St})$ -labeled St-tree $T \triangleq \langle T, u \rangle$ such that T is the set of histories h of \mathcal{G} starting with s and $u(h) \triangleq (f, q)$ where q is the last symbol in h and $f : \text{free}(\varphi) \rightarrow \text{Ac}$ is defined by $f(e) \triangleq \chi(e)(h)$ for all $e \in \text{free}(\varphi)$.¹*

The following lemma is proved by induction on the structure of the formula φ , as in [24]. The idea for handling the new case, i.e., the graded quantifier $\langle \langle x_1, \dots, x_n \rangle \rangle^{\geq g} \psi$, is to build an APT that is a projection of an APT that itself checks that each of the g tuples of strategies satisfies ψ and that each pair of g tuples is distinct.

Lemma 4.1 *For every GRADED-SL formula φ , CGS \mathcal{G} , and state $s \in \text{St}$, there exists an APT \mathcal{A}_φ such that for all assignments χ , if T is the encoding of χ (w.r.t. φ, \mathcal{G}, s), then $\mathcal{G}, \chi, s \models \varphi$ iff $T \in L(\mathcal{A}_\varphi)$.*

We make some remarks about this lemma. First, all the cases in the induction incur a linear blowup except for the quantification case (recall that the translation from an APT to an NPT results in an exponentially larger automaton [20]). Thus, the size of the APT for φ is non-elementary in the quantifier-rank of φ . However, we can say a little more. Note that a block of k identical quantifiers only costs, in the worst case, a single exponential (and not k many exponentials) because we can extend the proof above to deal with a block of quantifiers at once. Thus, we get that the size of the APT for φ is non-elementary in the quantifier-block rank of φ .

Theorem 4.1 *The model-checking problem for GRADED-SL is PTIME-COMPLETE w.r.t. the size of the model and $(k+1)\text{EXPTIME}$ if $k \geq 1$ is the quantifier-block rank of φ . Moreover, if φ is the form $\wp\psi$, where \wp is a quantifier-block, and ψ is of quantifier-block rank $k-1$, then the complexity is $k\text{EXPTIME}$.*

Proof. The lower-bound w.r.t the size of the model already holds for SL [24]. For the upper bound, use Lemma 4.1 to transform the CGS and φ into an APT and test its emptiness. The complexity of checking emptiness (or indeed, universality) of an APT is in EXPTIME [20]. As discussed after Lemma 4.1, the size of the APT is a tower of exponentials whose height is the quantifier-block rank of φ . This gives the $(k+1)\text{EXPTIME}$ upper bound. \square

Theorem 4.2 *The model-checking problem for GRADED-SL[NG] is PTIME-COMPLETE w.r.t. the size of the model and $(k+1)\text{-EXPTIME}$ when restricted to formulas of maximum alternation number k .*

Proof. The lower bound already holds for SL[NG] [24], and the upper bound is obtained by following the same reasoning for SL[NG] of the singleton existential quantifier [24] but using the automaton construction as in Theorem 4.1. \square

Directly from the statements reported above, we get the following results:

Theorem 4.3 *Checking the uniqueness of NE, and checking the uniqueness of SPE, can be done in 2EXPTIME .*

We conclude this section with the complexity of the model checking problem for GRADED-SL[1G]. Also in this case one can derive the lower bound from the one holding for the corresponding sub-logic in SL (SL[1G]) and the upper bound by using the same algorithm for SL[1G] but plugging a (yet no more

¹In case $\text{free}(\varphi) = \emptyset$, then f is the (unique) empty function. In this case, the encoding of every χ may be viewed as the tree-unwinding from s .

complex) different automata construction for the new existential quantifier modality. Indeed the model checking problem for GRADED-SL[1G] is 2EXPTIME-COMPLETE. It is worth recalling that SL[1G] strictly subsumes ATL* [24]. It is quite immediate to see that this also holds in the graded setting (note that ATL* already allows quantifying over tuples of agents' (bound) strategies). As the model checking for ATL* is already 2EXPTIME-HARD, we get that also for the graded extension for this logic, which we name GATL*, the model checking problem is 2EXPTIME-COMPLETE. The model checking results for both GATL* and GRADED-SL[1G] are reported in the following theorem.

Theorem 4.4 *The model-checking problem for GATL* and GRADED-SL[1G] is PTIME-COMPLETE w.r.t. the size of the model and 2-EXPTIME-COMPLETE in the size of formula.*

5 Conclusion

In this paper we introduced GRADED-SL to address and solve the unique NE problem. We have demonstrated that GRADED-SL is elegant, simple, and very powerful, and can solve the unique NE problem for LTL objectives in 2EXPTIME, and thus at the same complexity that is required to merely decide if a NE exists. We also instantiate our formalism by considering the well-known prisoner's dilemma and its iterated version. We have also shown that using the same approach one can express (and solve) the uniqueness of other standard solution concepts, e.g., subgame-perfect equilibria, again in 2EXPTIME. Finally, our work gives the first algorithmic solution to the model-checking problem of a graded variant of ATL*, and proves it to be 2EXPTIME-COMPLETE.

The positive results presented in this paper open several directions for future work. First, one may add other types of graded quantifiers, i.e., “there exists infinitely many strategies” and “there exists countably many strategies”, and still retain decidability using the results of [6]. Second, one may study the expressive power of SL with the atomic expression $x = y$ (saying that strategies x and y are the same), just as one does for first-order logic with and without equality. Third, one might extend fragments of GRADED-SL to quantitative objectives, such as mean-payoff or discounted-sum. Finally, one may implement GRADED-SL and its model-checking procedure in a formal verification tool such as SLK-MCMAS [10].

Acknowledgments

We thank Michael Wooldridge for suggesting uniqueness of Nash Equilibria as an application of graded strategy logic. Benjamin Aminof is supported by the Austrian National Research Network S11403-N23 (RiSE) of the Austrian Science Fund (FWF) and by the Vienna Science and Technology Fund (WWTF) through grant ICT12-059. Sasha Rubin is a Marie Curie fellow of the Istituto Nazionale di Alta Matematica. Aniello Murano is partially supported by the GNCS 2016 project: Logica, Automi e Giochi per Sistemi Auto-adattivi.

References

- [1] E. Altman, H. Kameda & Y. Hosokawa (2002): *Nash Equilibria in Load Balancing in Distributed Computer Systems*. *IGTR* 4(2), pp. 91–100.
- [2] R. Alur, T.A. Henzinger & O. Kupferman (2002): *Alternating-Time Temporal Logic*. *JACM* 49(5), pp. 672–713.

- [3] R. Alur, S. La Torre & P. Madhusudan (2003): *Playing Games with Boxes and Diamonds*. In: CONCUR 2003, LNCS 2761 2761, Springer, pp. 127–141.
- [4] B. Aminof, V. Malvone, A. Murano & S. Rubin (2016, (to appear)): *Graded Strategy Logic: Reasoning about Uniqueness of Nash Equilibria*. In: AAMAS’16, IFAAMAS.
- [5] R. Axelrod (1987): *The evolution of strategies in the iterated prisoners dilemma. The dynamics of norms*, pp. 1–16.
- [6] Vince Bárány, Lukasz Kaiser & Alexander Moshe Rabinovich (2010): *Expressing Cardinality Quantifiers in Monadic Second-Order Logic over Trees*. Fundam. Inform. 100(1-4), pp. 1–17.
- [7] K. G. Binmore (1992): *Fun and Games: A Text on Game Theory*. D.C. Heath.
- [8] P.A. Bonatti, C. Lutz, A. Murano & M.Y. Vardi (2008): *The Complexity of Enriched muCalculus*. LMCS 4(3), pp. 1–27.
- [9] D. Calvanese, G. De Giacomo & M. Lenzerini (1999): *Reasoning in Expressive Description Logics with Fixpoints based on Automata on Infinite Trees*. In: IJCAI 99, pp. 84–89.
- [10] P. Čermák, A. Lomuscio, F. Mogavero & A. Murano (2014): *MCMAS-SLK: A Model Checker for the Verification of Strategy Logic Specifications*. In: CAV’14, LNCS 8559, Springer, pp. 524–531.
- [11] K. Chatterjee, T.A. Henzinger & N. Piterman (2010): *Strategy Logic*. IC 208(6), pp. 677–693.
- [12] J. Bramel D. Simchi-Levi, X. Chen (2013): *The Logic of Logistics: Theory, Algorithms, and Applications for Logistics Management*. Science and Business Media, Springer.
- [13] E. A. Emerson & C. S. Jutla (1991): *Tree Automata, Mu-Calculus and Determinacy (Extended Abstract)*. In: ASFCs, pp. 368–377.
- [14] Clive D. Fraser (1992): *The uniqueness of Nash equilibrium in the private provision of public goods: an alternative proof*. Journal of Public Economics 49(3), pp. 389–390.
- [15] A. Glazer & K. A. Konrad (1993): *Private Provision of Public Goods, Limited Tax Deducibility, and Crowding Out*. FinanzArchiv / Public Finance Analysis 50(2), pp. 203–216.
- [16] Erich Grädel (1999): *On The Restraining Power of Guards*. J. Symb. Log. 64(4), pp. 1719–1742.
- [17] J. Gutierrez, P. Harrenstein & M. Wooldridge (2014): *Reasoning about Equilibria in Game-Like Concurrent Systems*. In: KR 14.
- [18] O. Kupferman, G. Perelli & M. Y. Vardi (2014): *Synthesis with Rational Environments*. In: EUMAS 2014, pp. 219–235.
- [19] O. Kupferman, U. Sattler & M.Y. Vardi (2002): *The Complexity of the Graded muCalculus*. In: CADE’02, LNCS 2392, Springer, pp. 423–437.
- [20] O. Kupferman, M.Y. Vardi & P. Wolper (2000): *An Automata Theoretic Approach to Branching-Time Model Checking*. JACM 47(2), pp. 312–360.
- [21] K. Leyton-Brown & Y. Shoham (2008): *Essentials of Game Theory: A Concise, Multidisciplinary Introduction (Synthesis Lectures on Artificial Intelligence and Machine Learning)*. M&C.
- [22] V. Malvone, F. Mogavero, A. Murano & L. Sorrentino (2015): *On the Counting of Strategies*. In: TIME 15 (To appear).
- [23] F. Mogavero, A. Murano, G. Perelli & M.Y. Vardi (2012): *What Makes ATL^{*} Decidable? A Decidable Fragment of Strategy Logic*. In: CONCUR’12, LNCS 7454, Springer, pp. 193–208.
- [24] F. Mogavero, A. Murano, G. Perelli & M.Y. Vardi (2014): *Reasoning About Strategies: On the Model-Checking Problem*. TOCL 15(4), pp. 34:1–42.
- [25] F. Mogavero, A. Murano & M.Y. Vardi (2010): *Reasoning About Strategies*. In: FSTTCS’10, LIPIcs 8, Leibniz-Zentrum fuer Informatik, pp. 133–144.
- [26] D. E. Muller & P. E. Schupp (1995): *Simulating Alternating Tree Automata by Nondeterministic Automata: New Results and New Proofs of the Theorems of Rabin, McNaughton and Safra*. Theor. Comput. Sci. 141(1&2), pp. 69–107.

- [27] A. Orda, R. Rom & N. Shimkin (1993): *Competitive routing in multiuser communication networks*. IEEE/ACM Trans. Netw. 1(5), pp. 510–521.
- [28] A. Pnueli & R. Rosner (1990): *Distributed reactive systems are hard to synthesize*. In: FOCS'90, IEEE, pp. 746–757.
- [29] Ariel Rubinstein (1986): *Finite automata play the repeated prisoner's dilemma*. Journal of Economic Theory 39(1), pp. 83–96.
- [30] R. Selten (1965): *Spieltheoretische Behandlung eines Oligopolmodells mit Nachfragetragheit*. Zeitschrift für die gesamte Staatswissenschaft 121, pp. 301–324.
- [31] Wolfgang Thomas (1990): *Infinite Trees and Automaton Definable Relations over Omega-Words*, pp. 263–277.
- [32] M. Ummels (2006): *Rational Behaviour and Strategy Construction in Infinite Multiplayer Games*. In: FSTTCS, pp. 212–223.

A Definition of Alternation Number

Definition A.1 The alternation number of a GRADED-SL[NG] formula is given by:

- $\text{alt}(p) \triangleq 0$, where $p \in \text{AP}$;
- $\text{alt}(\text{OP}\varphi) \triangleq \text{alt}(\varphi)$, where $\text{OP} \in \{\neg, \text{X}, \text{b}\}$;
- $\text{alt}(\varphi_1 \text{OP} \varphi_2) \triangleq \max(\text{alt}(\varphi_1), \text{alt}(\varphi_2))$ where $\text{OP} \in \{\vee, \text{U}\}$;
- $\text{alt}(\varphi\varphi) \triangleq \max(\text{alt}(\varphi), \text{alt}(\varphi))$ where φ is a quantification prefix;
- $\text{alt}(\varphi) \triangleq \sum_{i=1}^{|\varphi|-1} \text{switch}(\varphi_i, \varphi_{i+1})$, where $\text{switch}(Q, Q') = 1$ if Q and Q' are either both universal or both existential quantifiers, and 0 otherwise.

B Alternating Parity Tree Automata (APT)

For a set X , let $\text{B}^+(X)$ be the set of positive Boolean formulas over X , including the constants **true** and **false**. A set $Y \subseteq X$ satisfies a formula $\theta \in \text{B}^+(X)$, written $Y \models \theta$, if assigning **true** to elements in Y and **false** to elements in $X \setminus Y$ makes θ true.

Definition B.1 An Alternating Parity Tree-Automaton (APT) is a tuple $\mathcal{A} \triangleq \langle \Sigma, \Delta, Q, \delta, q_0, \mathfrak{X} \rangle$, where Σ is the input alphabet, Δ is a set of directions, Q is a finite set of states, $q_0 \in Q$ is an initial state, $\delta : Q \times \Sigma \rightarrow \text{B}^+(\Delta \times Q)$ is an alternating transition function, and \mathfrak{X} , an acceptance condition, is of the form $(F_1, \dots, F_k) \in (2^Q)^+$ where $F_1 \subseteq F_2 \dots \subseteq F_k = Q$.

The set $\Delta \times Q$ is called the set of *moves*. An NTA is an ATA in which each conjunction in the transition function δ has exactly one move (d, q) associated with each direction d .

An *input tree* for an APT is a Σ -labeled Δ -tree $T = \langle T, v \rangle$. A *run* of an APT on an input tree $T = \langle T, v \rangle$ is a $(\Delta \times Q)$ -tree R such that, for all nodes $x \in R$, where $x = (d_1, q_1) \dots (d_n, q_n)$ (for some $n \in \mathbb{N}$), it holds that (i) $y \triangleq (d_1, \dots, d_n) \in T$ and (ii) there is a set of moves $S \subseteq \Delta \times Q$ with $S \models \delta(q_n, v(y))$ such that $x \cdot (d, q) \in R$ for all $(d, q) \in S$.

The acceptance condition allows us to say when a run is successful. Let R be a run of an APT \mathcal{A} on an input tree T and $u \in (\Delta \times Q)^\omega$ one of its branches. Let $\text{inf}(u) \subseteq Q$ denote the set of states that occur in infinitely many moves of u . Say that u satisfies the parity acceptance condition $\mathfrak{X} = (F_1, \dots, F_k)$ if the least index $i \in [1, k]$ for which $\text{inf}(u) \cap F_i \neq \emptyset$ is even. A run is *successful* if all its branches satisfy the parity acceptance condition \mathfrak{X} . An APT *accepts* an input tree T iff there exists a successful run R of \mathcal{A} on T .

C Proof of Lemma 4.1.

As in [24] we induct on the structure of the formula φ to construct the corresponding automaton \mathcal{A}_φ . The Boolean operations are easily dealt with using the fact that disjunction corresponds to non-determinism, and negation corresponds to dualising the automaton. Note (\dagger) that thus also conjunction is dealt with due to De Morgan's laws. The temporal operators are dealt with by following the unique play (determined by the given assignment) and verifying the required subformulas, e.g., for $\text{X } \psi$ the automaton, after taking one step along the play, launches a copy of the automaton for ψ . All of these operations incur a linear blowup in the size of the automaton. The only case that differs from [24] is the quantification, i.e., we need to handle the case that $\varphi = \langle \langle x_1, \dots, x_n \rangle \rangle^{\geq g} \psi$. Recall that $\mathcal{G}, \chi, s \models \langle \langle x_1, \dots, x_n \rangle \rangle^{\geq g} \psi$ iff there exists g many tuples $\overline{\sigma_1}, \dots, \overline{\sigma_g}$ of strategies such that: $\overline{\sigma_a} \neq \overline{\sigma_b}$ for $a \neq b$, and $\mathcal{G}, \chi[\bar{x} \mapsto \overline{\sigma_i}], s \models \psi$ for $1 \leq i \leq g$.

We show how to build an NPT for φ that mimics this definition: it will be a projection of an APT \mathcal{D}_ψ , which itself is the intersection of two automata, one checking that each of the g tuples of strategies satisfies ψ , and the other checking that each pair of the g tuples of strategies is distinct.

In more detail, introduce a set of fresh variables $X \triangleq \{x_i^j \in \text{Vr} : i \leq n, j \leq g\}$, and consider the formulas ψ^j (for $j \leq g$) formed from ψ by renaming x_i (for $i \leq n$) to x_i^j . Define $\psi' \triangleq \wedge_{j \leq g} \psi^j$. Note that, by induction, each ψ^j has a corresponding APT, and thus, using the conjunction-case (\dagger) above, there is an APT \mathcal{B} for ψ' . Note that the input alphabet for \mathcal{B} is $(\text{free}(\psi') \rightarrow \text{Ac}) \times \text{St}$ and that $X \subseteq \text{free}(\psi')$.

On the other hand, let \mathcal{C} be an APT with input alphabet $(\text{free}(\psi') \rightarrow \text{Ac}) \times \text{St}$ that accepts a tree $T = \langle T, v \rangle$ if and only if for every $a \neq b \leq g$ there exists $i \leq n$ and $h \in T$ such that $v(h) = (f, q)$ and $f(x_i^a) \neq f(x_i^b)$.

Form the APT \mathcal{D}_ψ for the intersection of \mathcal{B} and \mathcal{C} .

Now, using the classic transformation [26], we remove alternation from the APT \mathcal{D}_ψ to get an equivalent NPT \mathcal{N} (note that this step costs an exponential). Finally, use the fact that NPTs are closed under projection (with no blowup) to get an NPT for the language $\text{proj}_X(L(\mathcal{N}))$ of trees that encode assignments χ satisfying φ .

For completeness we recall this last step. If L is a language of Σ -labeled trees with $\Sigma \triangleq A \rightarrow B$, and $X \subset A$, then the X -projection of L , written $\text{proj}_X(L)$, is the language of Σ' -labeled trees with $\Sigma' \triangleq A \setminus X \rightarrow B$ such that $T \triangleq \langle T, v \rangle \in \text{proj}_X(L)$ if and only if there exists an X -labeled tree $\langle T, w \rangle$ such that the language L contains the tree $\langle T, u \rangle$ where $u : T \rightarrow (A \rightarrow B)$ maps $t \in T$ to $v(t) \cup w(t)$. Now, if \mathcal{N} is an NPT with input alphabet $\Sigma \triangleq A \rightarrow B$, and if $X \subset A$, then there is an NPT with input alphabet $\Sigma' \triangleq A \setminus X \rightarrow B$ with language $\text{proj}_X(L(\mathcal{N}))$.

The proof that the construction is correct is immediate. \square

Graded Strategy Logic: Reasoning about Uniqueness of Nash Equilibria

Benjamin Aminof
Technische Universität Wien, Austria
benj@forsyte.tuwien.ac.at

Vadim Malvone, Aniello Murano, Sasha Rubin
Università degli Studi di Napoli Federico II, Italy
{vadim.malvone, anielo.murano,
sasha.rubin}@unina.it

ABSTRACT

Strategy Logic (SL) is a well established formalism for strategic reasoning in multi-agent systems. In a nutshell, SL is built over LTL and treats strategies as first-order objects that can be associated with agents by means of a binding operator. In this work we introduce Graded Strategy Logic (GRADED-SL), an extension of SL by graded quantifiers over tuples of strategy variables such as “there exist at least g different tuples (x_1, \dots, x_n) of strategies”. We study the model-checking problem of GRADED-SL and prove that it is no harder than for SL, i.e., it is non-elementary in the quantifier rank.

We show that GRADED-SL allows one to count the number of different strategy profiles that are Nash equilibria (NE), or subgame-perfect equilibria (SPE). By analyzing the structure of the specific formulas involved, we conclude that the important problems of checking for the existence of a unique NE or SPE can both be solved in 2EXPTIME, which is not harder than merely checking for the existence of such equilibria.

Categories and Subject Descriptors

I.2.11 [Artificial Intelligence]: Distributed Artificial Intelligence — *Multiagent Systems*

General Terms

Theory, Verification

Keywords

Strategic logics; Graded modalities; Nash equilibrium

1. INTRODUCTION

Strategy Logic (SL) is a powerful formalism for reasoning about strategies in multi-agent systems [43, 44]. Strategies tell an agent what to do — they are functions that prescribe an action based on the history. The fundamental idea in SL is to treat strategies as first-order object. A strategy x can be quantified existentially $\langle\!\langle x \rangle\!\rangle$ (read: there exists a strategy x) and universally $\llbracket x \rrbracket$ (read: for all strategies x). Furthermore, strategies are not intrinsically glued to specific agents: the *binding* operator (α, x) allows one to bind an

Appears in: *Proceedings of the 15th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2016), J. Thangarajah, K. Tuyls, C. Jonker, S. Marsella (eds.), May 9–13, 2016, Singapore.*

Copyright © 2016, International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org). All rights reserved.

agent α to the strategy x . SL strictly subsumes several other logics for strategic reasoning including the well known ATL and ATL* [3]. Being a very powerful logic, SL can directly express many solution concepts [9, 19, 32, 35, 43] among which that a strategy profile \bar{x} is a Nash equilibrium, and thus also the existence of a Nash equilibrium (NE).

Nash equilibrium is one of the most important concepts in game theory, forming the basis of much of the recent fundamental work in multi-agent decision making. A challenging and important aspect is to establish whether a game admits a *unique* NE [2, 20, 48]. This problem impacts on the predictive power of NE since, in case there are multiple equilibria, the outcome of the game cannot be uniquely pinned down [21, 49, 57]. Unfortunately, uniqueness has mainly been established either for special cost functions [2], or for very restrictive game topologies [47]. More specifically, uniqueness of NE in game theory is proved with procedures that are separated from the ones that check for its existence [2]; these procedures require various transformations of the best response functions of individual contributors [28, 29, 53], and there is no general theory that can be applied to different application areas [2].

In this paper, we address and solve the problem of expressing the uniqueness of certain solution concepts (and NE in particular) in a principled and elegant way. We introduce an extension of SL called GRADED-SL and study its model-checking problem. More specifically, we extend SL by replacing the quantification $\langle\!\langle x \rangle\!\rangle$ and $\llbracket x \rrbracket$ over strategy variables with *graded quantification over tuples of strategy variables*: $\langle\!\langle x_1, \dots, x_n \rangle\!\rangle^{\geq g}$ (read $\langle\!\langle x_1, \dots, x_n \rangle\!\rangle^{\geq g}$ as “there exist at least g different tuples (x_1, \dots, x_n) of strategies”) and its dual $\llbracket x_1, \dots, x_n \rrbracket^{\leq g}$ ($g \in \mathbb{N}$). Here, two strategies are different if they disagree on some history. That is, we count strategies syntactically as is usual in graded extensions of modal and description logics [6, 12, 15, 30, 36]. The key for expressing uniqueness of NE is the combination of quantifying over tuples (instead of singleton variables), and adding counting (in the form of graded modalities).

We address the model-checking problem for GRADED-SL and prove that it has the same complexity as SL, i.e., it is non-elementary in the nesting depth of quantifiers. In particular, we show that model checking GRADED-SL formulas with a nesting depth $k > 0$ of blocks of quantifiers (a block of quantifiers is a maximally-consecutive sequence of quantifiers of the same type, i.e., either all existential, or all universal) is in $(k+1)\text{EXPTIME}$, and that for the special case where the formula starts with a block of quantifiers, it is in $k\text{EXPTIME}$. Since many natural formulas contain a very small number of

quantifiers, the complexity of the model-checking problem is not as bad as it seems. Specifically, several solution concepts can be expressed as SL formulas with a small number of quantifiers [9, 19, 32, 35, 43]. Since the existence of a NE, and the fact that there is at most one NE, can be expressed in GRADED-SL using simple formulas we are able to conclude that the problem of checking the uniqueness of a NE can be solved in 2EXPTIME. Previously, it was only known that existence of NE can be checked in 2EXPTIME [32, 43], and indeed it is 2EXPTIME-complete [4, 32, 50]. Thus, GRADED-SL is the first logic that can solve the existence and uniqueness of NE (as well as many other solution concepts) in a uniform way.

SL has a few natural syntactic fragments, the most powerful of which is called Nested-Goal SL which can express NE [43]. Similarly, we define GRADED-SL *Nested-Goal*, a syntactic fragment of GRADED-SL. The Nested-Goal restriction encompasses formulas in a special prenex normal form with a particular nested temporal structure that restricts the application of both strategy quantifiers and agent bindings (see Section 2 for details). We show that the graded extension of Nested-Goal SL has the same model-checking complexity, i.e., non-elementary in the *alternation number* of the quantifiers appearing in the formula (the alternation number is, roughly speaking, the maximum number of existential/universal quantifier switches [43]). Since uniqueness of NE can be expressed in Nested-Goal GRADED-SL using alternation one, we get an alternative proof for checking the NE uniqueness in 2EXPTIME.

We exemplify our definition and our automata-theoretic algorithm for the model-checking problem with a large number of applications. In particular, we use it for reasoning about repeated one-shot games such as the *iterated prisoner’s dilemma* (IPD) [51]. The prisoner’s dilemma game is a popular metaphor for the problem of stable cooperation and has been widely used in several application domains [8]. In the classic definition, it consists of two players, each of them has an option to defect or collaborate. More involved is the IPD in which the process is repeated and one can model reactive strategies in continuous play. The IPD has become a standard model for the evolution of cooperative behaviour within a community of egoistic agents, frequently cited for implications in both sociology and biology (see [8]). Evaluating the existence of NE in an IPD and, even more, its uniqueness, is a very challenging and complicated question due to the rich set of strategies such a game can admit [8, 16, 17]. In particular, such infinite-duration games need to be supported by more complex solution concepts such as *subgame-perfect* equilibrium [27, 41, 55]. Thanks to GRADED-SL and the related model-checking result, we get that checking the uniqueness of a NE in an IPD can be solved in 2EXPTIME.

Related work. The importance of solution concepts, verifying a unique equilibrium, and the relationship with logics for strategic reasoning is discussed above. We now give some highlights from the long and active investigation of graded modalities in the formal verification community.

Graded modalities were first studied in modal logic [26] and then exported to the field of *knowledge representation* to allow quantitative bounds on the set of individuals satisfying a given property. Specifically, they were considered as *counting quantifiers* in first-order logics [31] and *number restrictions* in *description logics* [34]. *Graded μ -calculus*, in which immediate-successor accessible worlds are counted, was intro-

duced to reason about graded modal logic with fixed-point operators [36]. Recently, the notion of graded modalities was extended to count the number of paths in the branching-time temporal logic formulas CTL and CTL* [7, 10]. In the verification of reactive systems, we mention two orthogonal approaches: module checking for graded μ -calculus [6, 25] and an extension of ATL by graded path modalities [24].

The work closest to ours is [40]: also motivated by counting NE, it introduces a graded extension of SL, called GSL. In contrast with our work, GSL has a very intricate way of counting strategies: it makes use of a semantic definition of strategies being equivalent, and counting in which equivalent strategies are counted as a single strategy. While this approach has been proved to be sound and general, it heavily complicates the model-checking problem. Indeed, only a very weak fragment of GSL has been solved in [40] by exploiting an *ad hoc* solution that does not seem to be easily scalable to (all of) GSL. Precisely, the fragment investigated there is the vanilla restriction of the graded version of one-goal SL [42]. There is a common belief that the one-goal fragment is not powerful enough to express the existence of a Nash Equilibrium in concurrent games. The smallest fragment that is known to be able to represent this is the so called Boolean-goal Strategy Logic, whose graded extension (in the GSL sense) has no known solution.¹

Outline. The sequel of the paper is structured as follows. In Section 2 we introduce GRADED-SL and provide some preliminary related concepts. In Section 3 we address the model-checking problem for GRADED-SL and its fragments. We conclude with Section 4 in which we have a discussion and suggestions for future work.

2. GRADED STRATEGY LOGIC

In this section we introduce Graded Strategy Logic, which we call GRADED-SL for short.

2.1 Models

Sentences of GRADED-SL are interpreted over *concurrent game structures*, just as for ATL and SL [3, 43].

DEFINITION 2.1. A concurrent game structure (CGS) is a tuple $\mathcal{G} \triangleq \langle AP, Ag, Ac, St, s_I, \text{ap}, \text{tr} \rangle$, where AP, Ag, Ac, and St are the sets of atomic propositions, agents, actions and states, respectively, $s_I \in St$ is the initial state, and $\text{ap} : St \rightarrow 2^{AP}$ is the labeling function mapping each state to the set of atomic propositions true in that state. Let $Dc \triangleq Ag \rightarrow Ac$ be the set of decisions, i.e., functions describing the choice of an action by every agent. Then, $\text{tr} : Dc \rightarrow (St \rightarrow St)$ denotes the transition function mapping every decision $\delta \in Dc$ to a function $\text{tr}(\delta) : St \rightarrow St$.

We will usually take the set Ag of agents to be $\{\alpha_1, \dots, \alpha_n\}$.

A path (from s) is a finite or infinite non-empty sequence of states $s_1 s_2 \dots$ such that $s = s_1$ and for every i there exists a decision δ with $\text{tr}(\delta)(s_i) = s_{i+1}$. The set of paths starting

¹In [33] it has been shown that, in the restricted case of turn-based structures it is possible to express the existence of Nash equilibria in m^-ATL^* [45], a memory-full variant of ATL* (hence included in one-goal SL), but exponentially more succinct — and thus with a much more expensive model-checking algorithm. As also the authors in [33] state, it is not clear how to extend this result to the concurrent setting, even in the two player case.

with s is denoted $\text{Pth}(s)$. The set of finite paths from s , called the *histories (from s)*, is denoted $\text{Hst}(s)$. A *strategy (from s)* is a function $\sigma \in \text{Str}(s) \triangleq \text{Hst}(s) \rightarrow \text{Ac}$ that prescribes which action has to be performed given a certain history. We write $\text{Pth}, \text{Hst}, \text{Str}$ for the set of all paths, histories, and strategies (no matter where they start).

We use the standard notion of equality between strategies, [38], i.e., $\sigma_1 = \sigma_2$ iff for all $\rho \in \text{Hst}$, $\sigma_1(\rho) = \sigma_2(\rho)$. This extends to equality between two n -tuples of strategies in the natural way, i.e., coordinate-wise.

2.2 Syntax

We describe the syntax as well as related basic concepts. GRADED-SL extends SL by replacing the classic singleton strategy quantifiers $\langle\!\langle x \rangle\!\rangle$ and $\llbracket x \rrbracket$ with the graded (tupled) quantifiers $\langle\!\langle x_1, \dots, x_n \rangle\!\rangle^{\geq g}$ and $\llbracket x_1, \dots, x_n \rrbracket^{< g}$, respectively, where each x_i belongs to a countable set of variables Vr and $g \in \mathbb{N}$ is called the *degree* of the quantifier. Intuitively, these are read as “there exist at least g tuples of strategies (x_1, \dots, x_n) ” and “all but less than g many tuples of strategies”, respectively. The syntax (α, x) denotes a *binding* of the agent α to the strategy x . The syntax of GRADED-SL is:

DEFINITION 2.2. GRADED-SL formulas are built inductively by means of the following grammar, where $p \in \text{AP}$, $\alpha \in \text{Ag}$, $x, x_1, \dots, x_n \in \text{Vr}$ such that $x_i \neq x_j$ for $i \neq j$, and $g, n \in \mathbb{N}$:

$$\varphi := p \mid \neg\varphi \mid \varphi \vee \varphi \mid X\varphi \mid \varphi U \varphi \mid \langle\!\langle x_1, \dots, x_n \rangle\!\rangle^{\geq g} \varphi \mid (\alpha, x)\varphi.$$

Notation. For the rest of the paper, whenever we write $\langle\!\langle x_1, \dots, x_n \rangle\!\rangle^{\geq g}$ we also mean that $x_i \neq x_j$ for $i \neq j$.

Shorthands are derived as usual. Specifically, $\text{true} \triangleq p \vee \neg p$, $\text{false} \triangleq \neg \text{true}$, $F\varphi \triangleq \text{true} U \varphi$, and $G\varphi \triangleq \neg F \neg \varphi$. Moreover, the graded universal operator corresponds to the dual of the existential one, i.e., $\llbracket x_1, \dots, x_n \rrbracket^{< g} \varphi \triangleq \neg \langle\!\langle x_1, \dots, x_n \rangle\!\rangle^{\geq g} \neg \varphi$. A *placeholder* refers to an agent or a variable. In order to define the semantics, we first define the concept of free placeholders in a formula. Intuitively, an agent or variable is free in φ if it does not have a strategy associated with it (either by quantification or binding) but one is required in order to ascertain if φ is true or not. The definition mimics that for SL [43]. It is included here both for completeness and because it is important for defining the model-checking procedure, in particular for the encoding of strategies as trees (Definition 3.2).

DEFINITION 2.3. The set of free agents and free variables of a GRADED-SL formula φ is given by the function $\text{free} : \text{GRADED-SL} \rightarrow 2^{\text{Ag} \cup \text{Vr}}$ defined as follows:

- $\text{free}(p) \triangleq \emptyset$, where $p \in \text{AP}$;
- $\text{free}(\neg\varphi) \triangleq \text{free}(\varphi)$;
- $\text{free}(\varphi_1 \vee \varphi_2) \triangleq \text{free}(\varphi_1) \cup \text{free}(\varphi_2)$;
- $\text{free}(X\varphi) \triangleq \text{Ag} \cup \text{free}(\varphi)$;
- $\text{free}(\varphi_1 U \varphi_2) \triangleq \text{Ag} \cup \text{free}(\varphi_1) \cup \text{free}(\varphi_2)$;
- $\text{free}(\langle\!\langle x_1, \dots, x_n \rangle\!\rangle^{\geq g} \varphi) \triangleq \text{free}(\varphi) \setminus \{x_1, \dots, x_n\}$;
- $\text{free}((\alpha, x)\varphi) \triangleq \text{free}(\varphi)$, if $\alpha \notin \text{free}(\varphi)$, where $\alpha \in \text{Ag}$ and $x \in \text{Vr}$;
- $\text{free}((\alpha, x)\varphi) \triangleq (\text{free}(\varphi) \setminus \{\alpha\}) \cup \{x\}$, if $\alpha \in \text{free}(\varphi)$, where $\alpha \in \text{Ag}$ and $x \in \text{Vr}$.

A formula φ without free agents (resp., variables), i.e., with $\text{free}(\varphi) \cap \text{Ag} = \emptyset$ (resp., $\text{free}(\varphi) \cap \text{Vr} = \emptyset$), is called agent-closed (resp., variable-closed). If φ is both agent- and variable-closed, it is called a sentence.

Another important concept that characterizes the syntax of GRADED-SL is the *alternation number* of quantifiers, i.e., the maximum number of quantifier switches $\langle\!\langle \cdot \rangle\!\rangle [\cdot], [\cdot] \langle\!\langle \cdot \rangle\!\rangle$, $\langle\!\langle \cdot \rangle\!\rangle \neg \langle\!\langle \cdot \rangle\!\rangle$, or $[\cdot] \neg [\cdot]$ that binds a tuple of variables in a subformula that is not a sentence. We denote by $\text{alt}(\varphi)$ the alternation number of a GRADED-SL formula φ . The *quantifier rank* of φ is the maximum nesting of quantifiers in φ , e.g., $\langle\!\langle x_1, \dots, x_n \rangle\!\rangle^{\geq g} (\alpha_1, x_1) \dots (\alpha_n, x_n) \wedge_{i=1}^n (\langle\!\langle y \rangle\!\rangle (\alpha_i, y) \psi_i) \rightarrow \psi$ has quantifier rank 2 if each ψ_i is quantifier free. Moreover, a *quantifier-block* of φ is a maximally-consecutive sequence of quantifiers in φ of the same type (i.e., either all existential, or all universal). The *quantifier-block rank* of φ is exactly like the quantifier rank except that a quantifier block of j quantifiers contributes 1 instead of j to the count.

SL has a few natural syntactic fragments, the most powerful of which is called Nested-Goal SL. Similarly, we define GRADED-SL *Nested-Goal* (abbreviated GRADED-SL[NG]), as a syntactic fragment of GRADED-SL. As in Nested-Goal SL, in GRADED-SL[NG] we require that bindings and quantifications appear in exhaustive blocks. I.e., whenever there is a quantification over a variable in a formula ψ it is part of a consecutive sequence of quantifiers that covers all of the free variables that appear in ψ , and whenever an agent is bound to a strategy then it is part of a consecutive sequence of bindings of all agents to strategies. Finally, formulas with free agents are not allowed. To formalize GRADED-SL[NG] we first introduce some notions. A *quantification prefix* over a finite set $V \subseteq \text{Vr}$ of variables is a sequence $\varphi \in \{\langle\!\langle x_1, \dots, x_n \rangle\!\rangle^{\geq g}, \llbracket x_1, \dots, x_n \rrbracket^{< g} : x_1, \dots, x_n \in V \wedge g \in \mathbb{N}\}^*$ such that each $x \in V$ occurs exactly once in φ . A *binding prefix* is a sequence $b \in \{(a, x) : \alpha \in \text{Ag} \wedge x \in \text{Vr}\}^{|\text{Ag}|}$ such that each $\alpha \in \text{Ag}$ occurs exactly once in b . We denote the set of binding prefixes by B_n , and the set of quantification prefixes over V by $Qn(V)$.

DEFINITION 2.4. GRADED-SL[NG] formulas are built inductively using the following grammar, with $p \in \text{AP}$, $\varphi \in Qn(V)$ ($V \subseteq \text{Vr}$), and $b \in B_n$:

$$\varphi ::= p \mid \neg\varphi \mid \varphi \vee \varphi \mid X\varphi \mid \varphi U \varphi \mid \varphi \varphi \mid b\varphi,$$

where in the rule $\varphi\varphi$ we require that φ is agent-closed and $\varphi \in Qn(\text{free}(\varphi))$.

We conclude this subsection by introducing GRADED-SL[1G], another graded extension of fragment of SL, namely the one-goal sub-logic. As the name says, this fragment is obtained by restricting GRADED-SL[NG] to encompass formulas with just one nested goal. The importance of this fragment in SL stems from the fact that it strictly includes ATL* while maintaining the same complexity for both the model checking and the satisfiability problems, i.e. 2EXPTIME-COMPLETE [42,43]. However, it is commonly believed that Nash Equilibrium cannot be expressed in this fragment. The definition of GRADED-SL[1G] follows.

DEFINITION 2.5. GRADED-SL[1G] formulas are built inductively using the following grammar, with $p \in \text{AP}$, $\varphi \in Qn(V)$ ($V \subseteq \text{Vr}$), and $b \in B_n$:

$$\varphi ::= p \mid \neg\varphi \mid \varphi \vee \varphi \mid X\varphi \mid \varphi U \varphi \mid \varphi b\varphi,$$

with φ quantification prefix over $\text{free}(b\varphi)$.

2.3 Semantics

As for SL, the interpretation of a GRADED-SL formula requires a valuation of its free placeholders. This is done via *assignments (from s)*, i.e., functions $\chi \in \text{Asg}(s) \triangleq (\text{Vr} \cup \text{Ag}) \rightarrow \text{Str}(s)$ mapping variables/agents to strategies. We denote by $\chi[e \mapsto \sigma]$, with $e \in \text{Vr} \cup \text{Ag}$ and $\sigma \in \text{Str}(s)$, the assignment that differs from χ only in the fact that e maps to σ . Extend this definition to tuples: for $\bar{e} = (e_1, \dots, e_n)$ with $e_i \neq e_j$ for $i \neq j$, define $\chi[\bar{e} \mapsto \bar{\sigma}]$ to be the assignment that differs from χ only in the fact that e_i maps to σ_i (for each i).

Since assignments are total functions, each assignment from s determines a unique path from s , called a play:

DEFINITION 2.6. For an assignment $\chi \in \text{Asg}(s)$ the (χ, s) -play denotes the path $\pi \in \text{Pth}(s)$ such that for all $i \in \mathbb{N}$, it holds that $\pi_{i+1} = \text{tr}(\text{dc})(\pi_i)$, where $\text{dc}(\alpha) \triangleq \chi(\alpha)(\pi_{\leq i})$ for $\alpha \in \text{Ag}$. The function $\text{play} : \text{Asg} \times \text{St} \rightarrow \text{Pth}$, with $\text{dom}(\text{play}) \triangleq \{(\chi, s) : \chi \in \text{Asg}(s)\}$, maps (χ, s) to the (χ, s) -play $\text{play}(\chi, s) \in \text{Pth}(s)$.

The notation $\pi_{\leq i}$ (resp. $\pi_{< i}$) denotes the prefix of the sequence π of length i (resp. $i - 1$). Similarly, the notation π_i denotes the i th symbol of π . Thus, $\text{play}(\chi, s)_i$ is the i th state on the play determined by χ from s .

The following definition of χ_i says how to interpret an assignment χ starting from a point i along the play, i.e., for each placeholder e , take the action the strategy $\chi(e)$ would do if it were given the prefix of the play up to i followed by the current history.

DEFINITION 2.7. For $\chi \in \text{Asg}(s)$ and $i \in \mathbb{N}$, writing $\rho \triangleq \text{play}(\chi, s)_{\leq i}$ (the prefix of the play up to i) and $t \triangleq \text{play}(\chi, s)_i$ (the last state of ρ) define $\chi_i \in \text{Asg}(t)$ to be the assignment from t that maps $e \in \text{Vr} \cup \text{Ag}$ to the strategy that maps $h \in \text{Hst}(t)$ to the action $\chi(e)(\rho_{< i} \cdot h)$.

We now define the semantics of GRADED-SL. In particular, we define $\mathcal{G}, \chi, s \models \varphi$ and say that φ holds at s in \mathcal{G} under χ .

DEFINITION 2.8. Fix a CGS \mathcal{G} . For all states $s \in \text{St}$ and assignments $\chi \in \text{Asg}(s)$, the relation $\mathcal{G}, \chi, s \models \varphi$ is defined inductively on the structure of φ :

- $\mathcal{G}, \chi, s \models p$ iff $p \in \text{ap}(s)$;
- $\mathcal{G}, \chi, s \models \neg\varphi$ iff $\mathcal{G}, \chi, s \not\models \varphi$;
- $\mathcal{G}, \chi, s \models \varphi_1 \vee \varphi_2$ iff $\mathcal{G}, \chi, s \models \varphi_1$ or $\mathcal{G}, \chi, s \models \varphi_2$;
- $\mathcal{G}, \chi, s \models \mathsf{X}\varphi$ iff $\mathcal{G}, \chi_1, \text{play}(\chi, s)_1 \models \varphi$;
- $\mathcal{G}, \chi, s \models \varphi_1 \mathsf{U} \varphi_2$ iff there is an index $i \in \mathbb{N}$ such that $\mathcal{G}, \chi_i, \text{play}(\chi, s)_i \models \varphi_2$ and, for all indexes $j \in \mathbb{N}$ with $j < i$, it holds that $\mathcal{G}, \chi_j, \text{play}(\chi, s)_j \models \varphi_1$;
- $\mathcal{G}, \chi, s \models (\alpha, x)\varphi$ iff $\mathcal{G}, \chi[\alpha \mapsto \chi(x)], s \models \varphi$;
- $\mathcal{G}, \chi, s \models \langle\langle x_1, \dots, x_n \rangle\rangle^{\geq g} \varphi$ iff there exist g many tuples $\bar{\sigma}_1, \dots, \bar{\sigma}_g$ of strategies such that:
 - $\bar{\sigma}_i \neq \bar{\sigma}_j$ for $i \neq j$, and
 - $\mathcal{G}, \chi[\bar{x} \mapsto \bar{\sigma}_i], s \models \varphi$ for $1 \leq i \leq g$.

Intuitively, the existential quantifier $\langle\langle x_1, \dots, x_n \rangle\rangle^{\geq g} \varphi$ allows us to count the number of distinct tuples of strategies that satisfy φ .

As usual, if χ and χ' agree on $\text{free}(\varphi)$, then $\mathcal{G}, \chi, s \models \varphi$ if and only if $\mathcal{G}, \chi', s \models \varphi$, i.e., the truth of φ does not depend on the values the assignment takes on placeholders that are not free. Thus, for a sentence φ , we write $\mathcal{G} \models \varphi$ to mean that $\mathcal{G}, \chi, s_I \models \varphi$ for some (equivalently, for all) assignments χ , and where s_I is the initial state of \mathcal{G} .

Comparison with other logics. In the following we give the main intuitions relating GRADED-SL with SL and a *naive* fragment of GRADED-SL in which quantifiers are over single variables (and not tuple of variables).

GRADED-SL extends SL by replacing universal and existential strategy quantifiers $\langle\langle x \rangle\rangle$ and $\llbracket x \rrbracket$ with their graded versions over tuples of variables $\langle\langle x_1, \dots, x_n \rangle\rangle^{\geq g}$ and $\llbracket x_1, \dots, x_n \rrbracket^{< g}$. Grades allow one to count, which is not possible, a priori, in SL. On the other hand, every formula of SL has an equivalent formula of GRADED-SL: formed by replacing every quantifier $\langle\langle x \rangle\rangle$ with $\langle\langle x \rangle\rangle^{\geq 1}$.

An important power of GRADED-SL is that it can quantify over tuples of strategy variables. Consider the formula $\varphi = \langle\langle x, y \rangle\rangle^{\geq 2}(a, x)(b, y)\psi$ in GRADED-SL that represents the property in which there exist two tuples of strategies that satisfy ψ , and compare it to the following attempt at expressing φ only quantifying over single strategies: $\varphi' \triangleq \langle\langle x \rangle\rangle^{\geq 2}\langle\langle y \rangle\rangle^{\geq 2}(a, x)(b, y)\psi$. Observe that φ' says that there are two different strategies σ_1 and σ_2 for agent α_1 , and for each σ_i there are two different strategies δ_1^i and δ_2^i for the agent α_2 that satisfy ψ . Thus, there are four tuples of strategies that satisfy ψ , i.e., $\{\sigma_1, \delta_1^1\}, \{\sigma_1, \delta_2^1\}, \{\sigma_2, \delta_1^2\}, \{\sigma_2, \delta_2^2\}$. Other attempts (such as $\langle\langle x \rangle\rangle^{\geq 2}\langle\langle y \rangle\rangle^{\geq 1}(a, x)(b, y)\psi$), also fail to capture φ as they restrict one of the agents to a single strategy. This demonstrates the inadequacy of quantifying over single strategies for counting solution concepts.

2.4 Games with temporal objectives

In game theory, players have objectives that are summarised in a payoff function they receive depending on the resulting play. In order to specify such payoffs we follow a formalisation from [35] called *objective* LTL. This will allow us to model the Prisoner’s Dilemma (PD), probably the most famous game in game-theory, as well as an iterated version (IPD). We then discuss appropriate solution concepts, and show how to express these in GRADED-SL.

Let \mathcal{G} be a CGS with n agents. Let $m \in \mathbb{N}$ and fix, for each agent $\alpha_i \in \text{Ag}$, an *objective* tuple $S_i \triangleq \langle f_i, \varphi_i^1, \dots, \varphi_i^m \rangle$, where $f_i : \{0, 1\}^m \rightarrow \mathbb{N}$, and each φ_i^j is an LTL formula over AP. If π is a play, then agent α_i receives payoff $f_i(\bar{h}) \in \mathbb{N}$ where the j ’th bit \bar{h}_j of \bar{h} is 1 if and only if $\pi \models \varphi_i^j$. For instance, f may count the number of formulas that are true.

Prisoner’s Dilemma – One shot and Iterated

Two people have been arrested for robbing a bank and placed in separate isolation cells. Each has two possible choices, remaining silent or confessing. If a robber confesses and the other remains silent, the former is released and the latter stays in prison for a long time. If both confess they get two convictions, but they will get early parole. If both remain silent, they get a lighter sentence (e.g., on firearms possession charges). The dilemma faced by the prisoners is that, whatever the choice of the other prisoner, each is better off confessing than remaining silent. But the result obtained when both confess is worse than if they both remain silent.

We describe this (one-shot) scenario with the CGS in Figure 1 and, for agent α_i , the objective $S_i \triangleq \langle f_i, \varphi_i^1, \varphi_i^2, \varphi_i^3, \varphi_i^4 \rangle$

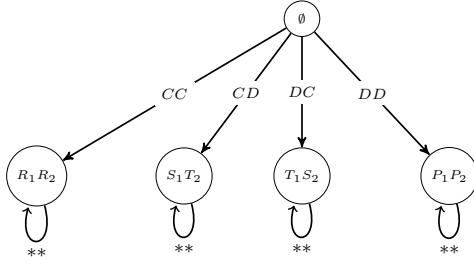


Figure 1: Prisoner’s dilemma.

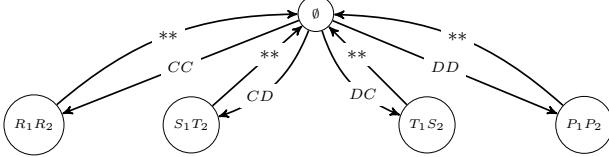


Figure 2: Iterated Prisoner’s dilemma.

where $\varphi_i^1 \triangleq \mathbf{X} \mathbf{S}_i$, $\varphi_i^2 \triangleq \mathbf{X} \mathbf{P}_i$, $\varphi_i^3 \triangleq \mathbf{X} \mathbf{R}_i$, and $\varphi_i^4 \triangleq \mathbf{X} \mathbf{T}_i$ and f_i returns the value of its input vector interpreted as a binary number, e.g., $f_i(0100) = 4$.

In words, we have two agents α_1 and α_2 . Each agent has two actions: cooperate (C) and defect (D), corresponding, respectively, to the options of remaining silent or confessing. For each possible pair of moves, the game goes in a state whose atomic propositions represent the payoffs: \mathbf{R}_i represents the *reward* payoff that α_i receives if both cooperate; \mathbf{P}_i is the *punishment* that α_i receives if both defect; \mathbf{T}_i is the *temptation* that α_i receives as a sole defector, and \mathbf{S}_i is the *sucker* payoff that α_i receives as a sole cooperator. The payoffs satisfy the following chain of inequalities: $\mathbf{T}_i > \mathbf{R}_i > \mathbf{P}_i > \mathbf{S}_i$.

The Iterated Prisoner’s Dilemma (IPD) is used to model a series of interactions. This is like PD except that after the first choice, both agents have another choice and so on. This is modeled in Figure 2. The main difference between the one-shot and iterated PD is that in the latter the agents’ actions (may) depend on the past behaviour. Convenient payoff functions for the IPD are the mean-average and discounted-payoff [11]. Such quantitative payoffs can be replaced, in a first approximation, by LTL payoffs such as “maximise the largest payoff I receive infinitely often”. This is formalised, for agent α_i , by the objective $S_i \triangleq \langle f_i, \varphi_i^1, \varphi_i^2, \varphi_i^3, \varphi_i^4 \rangle$ where $\varphi_i^1 \triangleq \mathbf{G} \mathbf{F} \mathbf{S}_i$, $\varphi_i^2 \triangleq \mathbf{G} \mathbf{F} \mathbf{P}_i$, $\varphi_i^3 \triangleq \mathbf{G} \mathbf{F} \mathbf{R}_i$, and $\varphi_i^4 \triangleq \mathbf{G} \mathbf{F} \mathbf{T}_i$, and f_i is as in the one-shot PD.

Solution Concepts. Solution concepts are criteria by which one captures what rational agents would do. This is especially relevant in case each agent has its own objective. The central solution concept in game theory is the Nash Equilibrium.

A tuple of strategies, one for each player, is called a *strategy profile*. A strategy profile is a *Nash equilibrium (NE)* if no agent can increase his payoff by unilaterally choosing a different strategy. A game may have zero, one, or many NE.

Consider first a CGS \mathcal{G} with n agents, where the objective of the agent $\alpha_i \in \text{Ag}$ contains a single LTL formula φ_i (with a larger payoff if it holds than if it doesn’t). It is not hard to see that the following formula of SL expresses that

$\bar{x} \triangleq (x_1 \dots x_n)$ is a Nash Equilibrium:

$$\psi_{NE}^1(\bar{x}) \triangleq (\alpha_1, x_1) \dots (\alpha_n, x_n) \bigwedge_{i=1}^n (\langle\langle y \rangle\rangle(\alpha_i, y) \varphi_i) \rightarrow \varphi_i$$

An alternative (which we will later use to obtain a formula in the fragment GRADED-SL[NG] that expresses the existence of a unique NE) is the following:

$$\phi_{NE}^1(\bar{x}) \triangleq [\![y_1]\!] \dots [\![y_n]\!] \bigwedge_{i=1}^n (\flat_i \varphi_i) \rightarrow \flat \varphi_i$$

where $\flat = (\alpha_1, x_1) \dots (\alpha_n, x_n)$, and $\flat_i = (\alpha_1, x_1) \dots (\alpha_{i-1}, x_{i-1}) (\alpha_i, y_i) (\alpha_{i+1}, x_{i+1}) \dots (\alpha_n, x_n)$.

Consider now the general case, where each agent α_i has an objective tuple $S_i \triangleq \langle f_i, \varphi_i^1, \dots, \varphi_i^m \rangle$. Given a vector $\bar{h} \in \{0, 1\}^m$, let $gd_i(\bar{h}) \triangleq \{\bar{t} \in \{0, 1\}^m \mid f_i(\bar{t}) \geq f_i(\bar{h})\}$ be the set of vectors \bar{t} for which the payoff for agent α_i is not worse than for \bar{h} . Also, let $\eta_i^{\bar{h}}$ be the formula obtained by taking a conjunction of the formulas $\varphi_i^1, \dots, \varphi_i^m$ or their negations according to \bar{h} , i.e., by taking φ_a^j if the j ’th bit in \bar{h} is 1, and otherwise taking $\neg \varphi_a^j$. Formally, $\eta_i^{\bar{h}} \triangleq \bigwedge_{j \in \{1 \leq j \leq m \mid h_j = 1\}} \varphi_i^j \wedge \bigwedge_{j \in \{1 \leq j \leq m \mid h_j = 0\}} \neg \varphi_i^j$. Observe that the following formula says that $\bar{x} \triangleq (x_1 \dots x_n)$ is a Nash Equilibrium:

$$\begin{aligned} \psi_{NE}(\bar{x}) &\triangleq (\alpha_1, x_1) \dots (\alpha_n, x_n) \\ &\bigwedge_{i=1}^n \bigwedge_{\bar{h} \in \{0, 1\}^m} (\langle\langle y \rangle\rangle(\alpha_i, y) \eta_i^{\bar{h}}) \rightarrow \bigvee_{\bar{t} \in gd_i(\bar{h})} \eta_i^{\bar{t}} \end{aligned}$$

Alike, one can modify $\phi_{NE}^1(\bar{x})$ to obtain a similar formula $\phi_{NE}(\bar{x})$ expressing that $\bar{x} \triangleq (x_1 \dots x_n)$ is a Nash Equilibrium:

$$\begin{aligned} \phi_{NE}(\bar{x}) &\triangleq [\![y_1]\!] \dots [\![y_n]\!] \\ &\bigwedge_{i=1}^n \bigwedge_{\bar{h} \in \{0, 1\}^m} (\flat_i \eta_i^{\bar{h}}) \rightarrow \bigvee_{\bar{t} \in gd_i(\bar{h})} \flat \eta_i^{\bar{t}} \end{aligned}$$

Going back to the PD example, due to the simplicity of the payoff functions, the formula ψ_{NE} collapses to become:

$$\psi_{PD}(\bar{x}) \triangleq (\alpha_1, x_1) \dots (\alpha_n, x_n) \bigwedge_{i=1}^2 \bigwedge_{j=1}^4 (\langle\langle y \rangle\rangle(\alpha_i, y) \varphi_i^j) \Rightarrow (\bigvee_{r \geq j} \varphi_i^r)$$

As it turns out (again due to the simplicity of the payoff functions), the formula above is also correct for the IPD.

It has been argued (in [35, 55]) that NE may be implausible when used for sequential games (of which iterated one shot games are central examples), and that a more robust notion is subgame-perfect equilibrium [52]. Given a game \mathcal{G} , a strategy profile is a *subgame-perfect equilibrium (SPE)* if for every possible history of the game, the strategies are an NE. The following formula expresses that $\bar{x} \triangleq (\alpha_1, x_1) \dots (\alpha_n, x_n)$ is an SPE:

$$\phi_{SPE}(\bar{x}) \triangleq [\![z_1, \dots, z_n]\!](\alpha_1, z_1) \dots (\alpha_n, z_n) \mathbf{G} \phi_{NE}(\bar{x})$$

Using graded modalities, we can thus express the uniqueness of a NE using the following GRADED-SL formula:

$$\langle\langle x_1, \dots, x_n \rangle\rangle^{\geq 1} \psi_{NE}(\bar{x}) \wedge \neg \langle\langle x_1, \dots, x_n \rangle\rangle^{\geq 2} \psi_{NE}(\bar{x})$$

By replacing ψ_{NE} with ϕ_{NE} (resp. by ϕ_{SPE}) in the formula above, we can express the uniqueness of a NE (resp. SPE) in GRADED-SL[NG].

3. THE MODEL-CHECKING PROCEDURE

In this section we study the model-checking problem for GRADED-SL and show that it is decidable with a time-complexity that is non-elementary (i.e., not bounded by any fixed tower of exponentials). However, it is elementary if the number of blocks of quantifiers is fixed. For the algorithmic procedures, we follow an *automata-theoretic approach* [37], reducing the decision problem for the logic to the emptiness problem of an automaton. The procedure we propose here extends that used for SL in [43]. The only case that is different is the new graded quantifier over tuples of strategies.

We start with the central notions of automata theory, and then show how to convert a GRADED-SL sentence φ into an automaton that accepts exactly the (tree encodings) of the concurrent game structures that satisfy φ . This is used to prove the main result about GRADED-SL model checking.

3.1 Automata Theory

A Σ -labeled Υ -tree T is a pair $\langle T, V \rangle$ where $T \subseteq \Upsilon^+$ is prefix-closed (i.e., if $t \in T$ and $s \in \Upsilon^+$ is a prefix of t then also $s \in T$), and $V : T \rightarrow \Sigma$ is a labeling function. Note that every word $w \in \Upsilon^+ \cup \Upsilon^\omega$ with the property that every prefix of w is in T , can be thought of as a path in T . Infinite paths are called *branches*.

Nondeterministic tree automata (NTA) are a generalization to infinite trees of the classical automata on words [54]. *Alternating tree automata* (ATA) are a further generalization of nondeterministic tree automata [23]. Intuitively, on visiting a node of the input tree, while an NTA sends exactly one copy of itself to each of the successors of the node, an ATA can send several copies to the same successor. We use the parity acceptance condition [37].

For a set X , let $\mathcal{B}^+(X)$ be the set of positive Boolean formulas over X , including the constants **true** and **false**. A set $Y \subseteq X$ satisfies a formula $\theta \in \mathcal{B}^+(X)$, written $Y \models \theta$, if assigning **true** to elements in Y and **false** to elements in $X \setminus Y$ makes θ true.

DEFINITION 3.1. An Alternating Parity Tree-Automaton (APT) is a tuple $\mathcal{A} \triangleq \langle \Sigma, \Delta, Q, \delta, q_0, \aleph \rangle$, where Σ is the input alphabet, Δ is a set of directions, Q is a finite set of states, $q_0 \in Q$ is an initial state, $\delta : Q \times \Sigma \rightarrow \mathcal{B}^+(\Delta \times Q)$ is an alternating transition function, and \aleph , an acceptance condition, is of the form $(F_1, \dots, F_k) \in (2^Q)^+$ where $F_1 \subseteq F_2 \dots \subseteq F_k = Q$.

The set $\Delta \times Q$ is called the set of *moves*. An NTA is an ATA in which each conjunction in the transition function δ has exactly one move (d, q) associated with each direction d .

An *input tree* for an APT is a Σ -labeled Δ -tree $T = \langle T, v \rangle$. A *run* of an APT on an input tree $T = \langle T, v \rangle$ is a $(\Delta \times Q)$ -tree R such that, for all nodes $x \in R$, where $x = (d_1, q_1) \dots (d_n, q_n)$ (for some $n \in \mathbb{N}$), it holds that (i) $y \triangleq (d_1, \dots, d_n) \in T$ and (ii) there is a set of moves $S \subseteq \Delta \times Q$ with $S \models \delta(q_n, v(y))$ such that $x \cdot (d, q) \in R$ for all $(d, q) \in S$.

The acceptance condition allows us to say when a run is successful. Let R be a run of an APT \mathcal{A} on an input tree T and $u \in (\Delta \times Q)^\omega$ one of its branches. Let $\text{inf}(u) \subseteq Q$ denote the set of states that occur in infinitely many moves of u . Say that u satisfies the parity acceptance condition $\aleph = (F_1, \dots, F_k)$ if the least index $i \in [1, k]$ for which $\text{inf}(u) \cap F_i \neq \emptyset$ is even. An APT *accepts* an input tree T iff there exists a run R of \mathcal{A} on T such that all its branches satisfy the acceptance condition

\aleph . The *language* $L(\mathcal{A})$ of the APT \mathcal{A} is the set of trees T accepted by \mathcal{A} . Two automata are *equivalent* if they have the same language. The *emptiness problem* for alternating parity tree-automata is to decide, given \mathcal{A} , whether $L(\mathcal{A}) = \emptyset$. The *universality problem* is to decide whether \mathcal{A} accepts all trees.

3.2 From Logic to Automata

Following an automata-theoretic approach, we reduce the model-checking problem of GRADED-SL to the emptiness problem for alternating parity tree automata [43]. The main step is to translate every GRADED-SL formula φ (i.e., φ may have free placeholders), concurrent-game structure \mathcal{G} , and state s , into an APT that accepts a tree if and only if the tree encodes an assignment χ such that $\mathcal{G}, \chi, s \models \varphi$.

We first describe the encoding, following [43]. Informally, the CGS \mathcal{G} is encoded by its “tree-unwinding starting from s ” whose nodes represent histories, i.e., the St-labeled St-tree $T \triangleq \langle \text{Hst}(s), u \rangle$ such that $u(h)$ is the last symbol of h . Then, every strategy $\chi(e)$ with $e \in \text{free}(\varphi)$ is encoded as an Ac-labelled tree over the unwinding. The unwinding and these strategies $\chi(e)$ are viewed as a single $(\text{VAL} \times \text{St})$ -labeled tree where $\text{VAL} \triangleq \text{free}(\varphi) \rightarrow \text{Ac}$.

DEFINITION 3.2. The encoding of χ (w.r.t. φ, \mathcal{G}, s) is the $(\text{VAL} \times \text{St})$ -labeled St-tree $T \triangleq \langle T, u \rangle$ such that T is the set of histories h of \mathcal{G} starting with s and $u(h) \triangleq (f, q)$ where q is the last symbol in h and $f : \text{free}(\varphi) \rightarrow \text{Ac}$ is defined by $f(e) \triangleq \chi(e)(h)$ for all $e \in \text{free}(\varphi)$.²

LEMMA 3.1. For every GRADED-SL formula φ , CGS \mathcal{G} , and state $s \in \text{St}$, there exists an APT \mathcal{A}_φ such that for all assignments χ , if T is the encoding of χ (w.r.t. φ, \mathcal{G}, s), then $\mathcal{G}, \chi, s \models \varphi$ iff $T \in L(\mathcal{A}_\varphi)$.

PROOF. As in [43] we induct on the structure of the formula φ to construct the corresponding automaton \mathcal{A}_φ . The Boolean operations are easily dealt with using the fact that disjunction corresponds to non-determinism, and negation corresponds to dualising the automaton. Note (\dagger) that thus also conjunction is dealt with due to De Morgan’s laws. The temporal operators are dealt with by following the unique play (determined by the given assignment) and verifying the required subformulas, e.g., for $\mathbf{X}\psi$ the automaton, after taking one step along the play, launches a copy of the automaton for ψ . All of these operations incur a linear blowup in the size of the automaton. The only case that differs from [43] is the quantification, i.e., we need to handle the case that $\varphi = \langle\!\langle x_1, \dots, x_n \rangle\!\rangle^{\geq g} \psi$. Recall that $\mathcal{G}, \chi, s \models \langle\!\langle x_1, \dots, x_n \rangle\!\rangle^{\geq g} \psi$ iff there exists g many tuples $\overline{\sigma_1}, \dots, \overline{\sigma_g}$ of strategies such that: $\overline{\sigma_a} \neq \overline{\sigma_b}$ for $a \neq b$, and $\mathcal{G}, \chi[\overline{x} \mapsto \overline{\sigma_i}], s \models \psi$ for $1 \leq i \leq g$. We show how to build an NPT for φ that mimics this definition: it will be a projection of an APT \mathcal{D}_ψ , which itself is the intersection of two automata, one checking that each of the g tuples of strategies satisfies ψ , and the other checking that each pair of the g tuples of strategies is distinct.

In more detail, introduce a set of fresh variables $X \triangleq \{x_i^j \in \text{Vr} : i \leq n, j \leq g\}$, and consider the formulas ψ^j (for $j \leq g$) formed from ψ by renaming x_i (for $i \leq n$) to x_i^j . Define $\psi' \triangleq \bigwedge_{j \leq g} \psi^j$. Note that, by induction, each ψ^j has a corresponding APT, and thus, using the conjunction-case

²In case $\text{free}(\varphi) = \emptyset$, then f is the (unique) empty function. In this case, the encoding of every χ may be viewed as the tree-unwinding from s .

(†) above, there is an APT \mathcal{B} for ψ' . Note that the input alphabet for \mathcal{B} is $(\text{free}(\psi') \rightarrow \text{Ac}) \times \text{St}$ and that $X \subseteq \text{free}(\psi')$.

On the other hand, let \mathcal{C} be an APT with input alphabet $(\text{free}(\psi') \rightarrow \text{Ac}) \times \text{St}$ that accepts a tree $T = \langle T, v \rangle$ if and only if for every $a \neq b \leq g$ there exists $i \leq n$ and $h \in T$ such that $v(h) = (f, q)$ and $f(x_i^a) \neq f(x_i^b)$.

Form the APT \mathcal{D}_ψ for the intersection of \mathcal{B} and \mathcal{C} .

Now, using the classic transformation [46], we remove alternation from the APT \mathcal{D}_ψ to get an equivalent NPT \mathcal{N} (note that this step costs an exponential). Finally, use the fact that NPTs are closed under projection (with no blowup) to get an NPT for the language $\text{proj}_X(L(\mathcal{N}))$ of trees that encode assignments χ satisfying φ .

For completeness we recall this last step. If L is a language of Σ -labeled trees with $\Sigma \triangleq A \rightarrow B$, and $X \subset A$, then the X -projection of L , written $\text{proj}_X(L)$, is the language of Σ' -labeled trees with $\Sigma' \triangleq A \setminus X \rightarrow B$ such that $T \triangleq \langle T, v \rangle \in \text{proj}_X(L)$ if and only if there exists an X -labeled tree $\langle T, w \rangle$ such that the language L contains the tree $\langle T, u \rangle$ where $u : T \rightarrow (A \rightarrow B)$ maps $t \in T$ to $v(t) \cup w(t)$. Now, if \mathcal{N} is an NPT with input alphabet $\Sigma \triangleq A \rightarrow B$, and if $X \subset A$, then there is an NPT with input alphabet $\Sigma' \triangleq A \setminus X \rightarrow B$ with language $\text{proj}_X(L(\mathcal{N}))$.

The proof that the construction is correct is immediate. \square

We make some remarks about the proof. First, all the cases in the induction incur a linear blowup except for the quantification case (recall that the translation from an APT to an NPT results in an exponentially larger automaton [37]). Thus, the size of the APT for φ is non-elementary in the quantifier-rank of φ . However, we can say a little more. Note that a block of k identical quantifiers only costs, in the worst case, a single exponential (and not k many exponentials) because we can extend the proof above to deal with a block of quantifiers at once. Thus, we get that the size of the APT for φ is non-elementary in the quantifier-block rank of φ .

Here is the main decidability result.

THEOREM 3.1. *The model-checking problem for GRADED-SL is PTIME-COMPLETE w.r.t. the size of the model and $(k+1)\text{EXPTIME}$ if $k \geq 1$ is the quantifier-block rank of φ . Moreover, if φ is the form $\wp\psi$, where \wp is a quantifier-block, and ψ is of quantifier-block rank $k-1$, then the complexity is $k\text{EXPTIME}$.*

PROOF. The lower-bound w.r.t the size of the model already holds for SL [43]. For the upper bound, use Lemma 3.1 to transform the CGS and φ into an APT and test its emptiness. The complexity of checking emptiness (or indeed, universality) of an APT is in EXPTIME [37]. As discussed after the proof of the Lemma, the size of the APT is a tower of exponentials whose height is the quantifier-block rank of φ . This gives the $(k+1)\text{EXPTIME}$ upper bound.

Moreover, suppose that $\varphi = \wp\psi$ where \wp consists of, say, n existential quantifiers (resp. universal quantifiers). The quantifier-block rank of ψ is $k-1$. Moreover, in the proof of Lemma 3.1, the APT \mathcal{D}_ψ , whose size is non-elementary in $k-1$, has the property that it is non-empty (resp. universal) if and only if the CGS satisfies $\wp\psi$. Conclude that model checking $\wp\psi$ can be done in $k\text{EXPTIME}$. \square

THEOREM 3.2. *The model-checking problem for GRADED-SL[NG] is PTIME-COMPLETE w.r.t. the size of the model and $(k+1)\text{-EXPTIME}$ when restricted to formulas of maximum alternation number k .*

PROOF. The lower bound already holds for SL[NG] [43], and the upper bound is obtained by following the same reasoning for SL[NG] of the singleton existential quantifier [43] but using the automaton construction as in Theorem 3.1. \square

Directly from the statements reported above, we get the following results:

THEOREM 3.3. *Checking the uniqueness of NE, and checking the uniqueness of SPE, can be done in 2EXPTIME.*

PROOF. For NE: by Section 2.4, we need to check that $\langle x_1, \dots, x_n \rangle^{\geq 1} \psi_{NE}(\bar{x})$ holds but $\langle x_1, \dots, x_n \rangle^{\geq 2} \psi_{NE}(\bar{x})$ does not; by the second part of Theorem 3.1, each of these two model-checking problems can be decided in 2EXPTIME.

For SPE: apply Theorem 3.2 and use the fact that the formula for SPE in Section 2.4 is in GRADED-SL Nested-Goal and has alternation number 1. \square

We conclude this section with the complexity of the model checking problem for GRADED-SL[1G]. Also in this case one can derive the lower bound from the one holding for the corresponding sub-logic in SL (SL[1G]) and the upper bound by using the same algorithm for SL[1G] but plugging a (yet no more complex) different automata construction for the new existential quantifier modality. Indeed the model checking problem for GRADED-SL[1G] is 2EXPTIME-COMPLETE. It is worth recalling that SL[1G] strictly subsumes ATL* [43]. It is quite immediate to see that this also holds in the graded setting (note that ATL* already allows quantifying over tuples of agents' (bound) strategies). As the model checking for ATL* is already 2EXPTIME-HARD, we get that also for the graded extension for this logic, which we name GATL*, the model checking problem is 2EXPTIME-COMPLETE. The model checking results for both GATL* and GRADED-SL[1G] are reported in the following theorem.

THEOREM 3.4. *The model-checking problem for GATL* and GRADED-SL[1G] is PTIME-COMPLETE w.r.t. the size of the model and 2-EXPTIME-COMPLETE in the size of formula.*

4. CONCLUSION

The Nash equilibrium is the foundational solution concept in game theory. The last twenty years have witnessed the introduction of many logical formalisms for modeling and reasoning about solution concepts, and NE in particular [9, 14, 19, 32, 39, 43, 44]. These formalisms are useful for addressing qualitative questions such as “*does the game admit a Nash equilibrium?*”. Among others, Strategy Logic (SL) has come to the fore as a general formalism that can express and solve this question, for LTL objectives, in 2EXPTIME. Contrast this with the fact that this question is 2EXPTIME-complete even for two player zero-sum LTL games [4].

One of the most important questions about NE in computational game theory is “*does the game admit more than one NE?*” [20, 48] — the unique NE problem. This problem is deeply investigated in game theory and is shown to be very challenging [2, 21, 28, 29, 47, 49, 53, 57]. Prior to this work, no logic-based technique, as far as we know, solved this problem.³ In this paper we introduced GRADED-SL to address

³In the related work section we discussed the logic GSL that, although motivated by the need to address the unique NE problem, only supplies a model-checking algorithm for a very small fragment of GSL that, it is assumed, is not able to express the existence of NE.

and solve the unique NE problem. We have demonstrated that GRADED-SL is elegant, simple, and very powerful, and can solve the unique NE problem for LTL objectives in 2EXPTIME, and thus at the same complexity that is required to merely decide if a NE exists. We also instantiate our formalism by considering the well-known prisoner’s dilemma and its iterated version. We have also shown that using the same approach one can express (and solve) the uniqueness of other standard solution concepts, e.g., subgame-perfect equilibria, again in 2EXPTIME. Finally, our work gives the first algorithmic solution to the model-checking problem of a graded variant of ATL*, and proves it to be 2EXPTIME-COMPLETE.

The positive results presented in this paper open several directions for future work. We are most excited about extending LTL objectives to quantitative objectives such as mean-payoff or discounted-payoff. These naturally extend classic games with quantitative aspects. That is, the result of a play is a real-valued payoff for each player [58]. In a mean-payoff game, one is interested in the *long-run average* of the edge-weights along a play, called the *value* of the play. In the basic setting, there are two players, one wishing to minimize this value, and the other to maximize it. In the discounted version, the weights associated with edges are “*discounted*” with time. In other words, an edge chosen at time t adds a weight to the long-run average that is greater than the value the same edge would contribute if chosen later on. Because of their applicability to economics these games have been studied from an algorithmic perspective for some time [58]. Also, the connection of mean-payoff and discounted-payoff objectives with NE has been recently investigated in the multi-agent setting (see [13] for a recent work). However, extending our results to the weighted setting may prove challenging since, in this setting, the automata-theoretic approach gives rise to weighted-automata, for which many problems are much harder or undecidable (though not in all cases) [1, 5].

In the multi-agent setting, reasoning about epistemic alternatives plays a key role. Thus, an important extension would be to combine the knowledge operators in SLK [18] with the graded quantifiers we introduced for GRADED-SL. Since strategic reasoning under imperfect information has an undecidable model-checking problem [22], one may restrict to memoryless strategies as was done for SLK. More involved, would be to add grades to the knowledge operators, thus being able to express “there exists at least g equivalent worlds” [56].

Last but not least, another direction is to consider implementing GRADED-SL and its model-checking procedure in a formal verification tool. A reasonable approach would be, for example, to extend the tool SLK-MCMAS [18].

Acknowledgments

We thank Michael Wooldridge for suggesting uniqueness of Nash Equilibria as an application of graded strategy logic. Benjamin Aminof is supported by the Austrian National Research Network S11403-N23 (RiSE) of the Austrian Science Fund (FWF) and by the Vienna Science and Technology Fund (WWTF) through grant ICT12-059. Sasha Rubin is a Marie Curie fellow of the Istituto Nazionale di Alta Matematica. Aniello Murano is partially supported by the GNCS 2016 project: Logica, Automi e Giochi per Sistemi Auto-adattivi.

REFERENCES

- [1] S. Almagor, U. Boker, and O. Kupferman. What’s decidable about weighted automata? In *ATVA’11*, LNCS 6996, pages 482–491, 2011.
- [2] E. Altman, H. Kameda, and Y. Hosokawa. Nash equilibria in load balancing in distributed computer systems. *IGTR*, 4(2):91–100, 2002.
- [3] R. Alur, T. Henzinger, and O. Kupferman. Alternating-Time Temporal Logic. *JACM*, 49(5):672–713, 2002.
- [4] R. Alur, S. La Torre, and P. Madhusudan. Playing games with boxes and diamonds. In *CONCUR’03*, LNCS 2761, pages 127–141. Springer, 2003.
- [5] B. Aminof, O. Kupferman, and R. Lampert. Rigorous approximated determinization of weighted automata. *Theor. Comput. Sci.*, 480:104–117, 2013.
- [6] B. Aminof, A. Legay, A. Murano, and O. Serre. μ -calculus pushdown module checking with imperfect state information. In *IFIP-TCS’08*, IFIP 273, pages 333–348. Springer, 2008.
- [7] B. Aminof, A. Murano, and S. Rubin. On CTL* with graded path modalities. In *LPAR-20’15*, LNCS 9450, pages 281–296, 2015.
- [8] R. Axelrod. The evolution of strategies in the iterated prisoners dilemma. *The dynamics of norms*, pages 1–16, 1987.
- [9] F. Belardinelli. A logic of knowledge and strategies with imperfect information. In *LAMAS’15*, 2015.
- [10] A. Bianco, F. Mogavero, and A. Murano. Graded Computation Tree Logic. *TOCL*, 13(3):25:1–53, 2012.
- [11] K. G. Binmore. *Fun and Games: A Text on Game Theory*. D.C. Heath, 1992.
- [12] P. Bonatti, C. Lutz, A. Murano, and M. Vardi. The Complexity of Enriched muCalculi. *LMCS*, 4(3):1–27, 2008.
- [13] E. Boros, K. M. Elbassioni, V. Gurvich, and K. Makino. Nested family of cyclic games with k-total effective rewards. *CoRR*, abs/1412.6072, 2014.
- [14] T. Brihaye, A. D. C. Lopes, F. Laroussinie, and N. Markey. ATL with Strategy Contexts and Bounded Memory. In *LFCS’09*, LNCS 5407, pages 92–106, 2009.
- [15] D. Calvanese, G. De Giacomo, and M. Lenzerini. Reasoning in expressive description logics with fixpoints based on automata on infinite trees. In *IJCAI’99*, pages 84–89, 1999.
- [16] V. Capraro. A model of human cooperation in social dilemmas. *CoRR*, abs/1307.4228, 2013.
- [17] V. Capraro, M. Venanzi, M. Polukarov, and N. R. Jennings. Cooperative equilibria in iterated social dilemmas. In *SAGT’13*, LNCS 8146, pages 146–158, 2013.
- [18] P. Čermák, A. Lomuscio, F. Mogavero, and A. Murano. MCMAS-SLK: A Model Checker for the Verification of Strategy Logic Specifications. In *CAV’14*, LNCS 8559, pages 524–531. Springer, 2014.
- [19] K. Chatterjee, T. Henzinger, and N. Piterman. Strategy Logic. *IC*, 208(6):677–693, 2010.
- [20] R. Cornes, R. Hartley, and T. Sandler. An elementary proof via contraction. *Journal of Public Economic Theory*, 1(4):499–509, 1999.
- [21] J. B. D. Simchi-Levi, X. Chen. *The Logic of Logistics*:

- Theory, Algorithms, and Applications for Logistics Management.* Science and Business Media. 2013.
- [22] C. Dima and F. Tiplea. Model-checking ATL under Imperfect Information and Perfect Recall Semantics is Undecidable. Technical report, arXiv, 2011.
- [23] E. A. Emerson and C. S. Jutla. Tree automata, mu-calculus and determinacy (extended abstract). In *ASFC'91*, pages 368–377, 1991.
- [24] M. Faella, M. Napoli, and M. Parente. Graded Alternating-Time Temporal Logic. *FI*, 105(1-2):189–210, 2010.
- [25] A. Ferrante, A. Murano, and M. Parente. Enriched Mu-Calculi Module Checking. *LMCS*, 4(3):1–21, 2008.
- [26] K. Fine. In So Many Possible Worlds. *NDJFL*, 13:516–520, 1972.
- [27] D. Fisman, O. Kupferman, and Y. Lustig. Rational Synthesis. In *TACAS'10*, LNCS 6015, pages 190–204. Springer, 2010.
- [28] C. D. Fraser. The uniqueness of nash equilibrium in the private provision of public goods: an alternative proof. *Journal of Public Economics*, 49(3):389–390, 1992.
- [29] A. Glazer and K. A. Konrad. Private provision of public goods, limited tax deducibility, and crowding out. *FinanzArchiv / Public Finance Analysis*, 50(2):203–216, 1993.
- [30] E. Grädel. On the restraining power of guards. *J. Symb. Log.*, 64(4):1719–1742, 1999.
- [31] E. Grädel, M. Otto, and E. Rosen. Two-Variable Logic with Counting is Decidable. In *LICS'97*, pages 306–317. IEEE Computer Society, 1997.
- [32] J. Gutierrez, P. Harrenstein, and M. Wooldridge. Reasoning about equilibria in game-like concurrent systems. In *KR'14*, 2014.
- [33] J. Gutierrez, P. Harrenstein, and M. Wooldridge. Expressiveness and complexity results for strategic reasoning. In *CONCUR'15*, LIPIcs 42, pages 268–282, 2015.
- [34] B. Hollunder and F. Baader. Qualifying Number Restrictions in Concept Languages. In *KR'91*, pages 335–346. Kaufmann, 1991.
- [35] O. Kupferman, G. Perelli, and M. Y. Vardi. Synthesis with rational environments. In *EUMAS'14*, LNCS 8953, pages 219–235, 2014.
- [36] O. Kupferman, U. Sattler, and M. Vardi. The Complexity of the Graded muCalculus. In *CADE'02*, LNCS 2392, pages 423–437. Springer, 2002.
- [37] O. Kupferman, M. Vardi, and P. Wolper. An Automata Theoretic Approach to Branching-Time Model Checking. *JACM*, 47(2):312–360, 2000.
- [38] K. Leyton-Brown and Y. Shoham. *Essentials of Game Theory: A Concise, Multidisciplinary Introduction (Synthesis Lectures on Artificial Intelligence and Machine Learning)*. M&C, 2008.
- [39] A. Lopes, F. Laroussinie, and N. Markey. ATL with Strategy Contexts: Expressiveness and Model Checking. In *FSTTCS'10*, LIPIcs 8, pages 120–132, 2010.
- [40] V. Malvone, F. Mogavero, A. Murano, and L. Sorrentino. On the counting of strategies. In *TIME'15*, pages 170–179, 2015.
- [41] J. H. Miller. The coevolution of automata in the repeated prisoner’s dilemma. *Journal of Economic Behavior & Organization*, 29(1):87–112, 1996.
- [42] F. Mogavero, A. Murano, G. Perelli, and M. Vardi. What Makes ATL* Decidable? A Decidable Fragment of Strategy Logic. In *CONCUR'12*, LNCS 7454, pages 193–208. Springer, 2012.
- [43] F. Mogavero, A. Murano, G. Perelli, and M. Vardi. Reasoning About Strategies: On the Model-Checking Problem. *TOCL*, 15(4):34:1–42, 2014.
- [44] F. Mogavero, A. Murano, and M. Vardi. Reasoning About Strategies. In *FSTTCS'10*, LIPIcs 8, pages 133–144. Leibniz-Zentrum fuer Informatik, 2010.
- [45] F. Mogavero, A. Murano, and M. Vardi. Relentful Strategic Reasoning in Alternating-Time Temporal Logic. In *LPAR'10*, LNAI 6355, pages 371–387, 2010.
- [46] D. E. Muller and P. E. Schupp. Simulating alternating tree automata by nondeterministic automata: New results and new proofs of the theorems of rabin, mcnaughton and safra. *Theor. Comput. Sci.*, 141(1&2):69–107, 1995.
- [47] A. Orda, R. Rom, and N. Shimkin. Competitive routing in multiuser communication networks. *IEEE/ACM Trans. Netw.*, 1(5):510–521, 1993.
- [48] G. Papavassiliopoulos and J. B. Cruz. On the uniqueness of nash strategies for a class of analytic differential games. *Journal of Optimization Theory and Applications*, 27(2):309–314, 1979.
- [49] L. Pavel. *Game Theory for Control of Optical Networks*. Science and Business Media. Springer, 2012.
- [50] A. Pnueli and R. Rosner. Distributed reactive systems are hard to synthesize. In *FOCS'90*, pages 746–757, 1990.
- [51] A. Rubinstein. Finite automata play the repeated prisoner’s dilemma. *Journal of Economic Theory*, 39(1):83–96, 1986.
- [52] R. Selten. Spieltheoretische behandlung eines oligopolmodells mit nachfragetragheit. *Zeitschrift fur die gesamte Staatswissenschaft*, 121:301–324, 1965.
- [53] H. R. V. T. C. Bergstrom, L. E. Blume. On the private provision of public goods. *Journal of Public Economics*, 29(1):25–49, 1986.
- [54] W. Thomas. Infinite trees and automaton definable relations over omega-words. In *STACS'90*, pages 263–277, 1990.
- [55] M. Ummels. Rational behaviour and strategy construction in infinite multiplayer games. In *FSTTCS'06*, LNCS 4337, pages 212–223, 2006.
- [56] W. van der Hoek and J.-J. Meyer. Graded modalities in epistemic logic. In *LFCS'92*, LNCS 620, pages 503–514. 1992.
- [57] Y. Zhang and M. Guizani. *Game Theory for Wireless Communications and Networking*. CRC Press, 2011.
- [58] U. Zwick and M. Paterson. The complexity of mean payoff games on graphs. *Theor. Comput. Sci.*, 158(1&2):343–359, 1996.

Automatic Verification of Multi-Agent Systems in Parameterised Grid-Environments

Benjamin Aminof
Technische Universität Wien,
Austria
benj@forsyte.at

Aniello Murano, Sasha Rubin
Università di Napoli Federico II, Italy
aniello.murano@unina.it
sasharubin@unina.it

Florian Zuleger
Technische Universität Wien,
Austria
zuleger@forsyte.at

ABSTRACT

We present a framework for modeling and analysing multiple mobile agents on grid-environments such as finite mazes and labyrinths. Agents are modeled as automata, and the grid-environments are parameterised by their size and the relative positions of the obstacles. We study the verification problem, i.e., whether given agents complete a given task on a given (possibly infinite) set of grid-environments. We identify restrictions on the agents and on the environments for which the verification problem is decidable (and in PSPACE). These assumptions are: i) there are a bounded number of obstacles, and ii) the agents are not allowed to issue commands like “increase my x -coordinate by 1” but can only issue commands that change their relative positions, e.g., “increase my x -coordinate until I go past this wall”.

We prove PSPACE-hardness already for the verification problem of a single agent on singleton parameterised environments with no obstacles. It is therefore remarkable that the PSPACE-upper bound also holds for the verification problem with multiple agents, parameterised environments and multiple obstacles. We prove that weakening either of restrictions i) or ii) results in undecidability. The importance of this work is that it is the first to give a sound and complete decision procedure for the verification problem on parameterised grid-like environments. Previous work either involved only a single grid, restricted the scheduling of the agents, or excluded grids altogether.

General Terms

Theory, Verification

Keywords

Computational Models; Autonomous Mobile Agents; Distributed Robot Systems; Grids; Parameterised Verification

1. INTRODUCTION

Physical multiagent systems are designed to move in space. Thus 2D or 3D space, or their abstractions into grids [41, 7, 25, 26, 3], are the canonical environments for modeling multiagent systems. In some cases, e.g., if the environment

Appears in: *Proceedings of the 15th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2016), J. Thangarajah, K. Tuyls, C. Jonker, S. Marsella (eds.), May 9–13, 2016, Singapore.*

Copyright © 2016, International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org). All rights reserved.

is too dangerous or expensive to be reached by humans, we may assume that agents initially only have partial information about the environment. For instance, they may not know the exact size of the space or the extent of the obstacles, but they may know the relative positions of obstacles. Thus, agents should be designed so that they operate correctly on all possible grids that are consistent with their information [44, 43, 7, 31, 19, 25, 3]. In this paper we study the verification problem for such multiagent systems that have partial information about the environment.

Model checking is one of the main paradigms for automatic verification: the system is modeled as a finite structure M , the specification is expressed as a formula ϕ in a suitable logical formalism, and a model checking algorithm is used to verify if the structure M satisfies the formula ϕ [13]. The main obstacle to using this paradigm in our setting is that, since the environment is not completely-known, we have to model check many (potentially infinitely many) structures M . There are two main approaches to overcoming this problem, abstraction [13] and parameterised model-checking [6]. We pursue the second approach.

Motivating Scenario: Navigating Manhattan. Suppose you are arrive at Grand Central Station (GCS) in midtown Manhattan and want to get to a certain landmark, say Carnegie Hall (CH), by walking along the streets and avenues. Perhaps you already know (because you have a crude map drawn on a piece of paper) the relative positions of some of the major landmarks,¹ e.g., Times Square (TS) is west of GCS, somewhere between TS and GCS is a point, North of which is the Museum of Modern Art (MOMA), and CH is North of MOMA. Can you use this information to reach CH? Any algorithm that you use to navigate will probably have primitives of the form “walk east until you reach landmark X”, or “walk east for a while, and stop at some point to look at your map again”. Now, suppose you have a generic algorithm for navigating, given the relative positions of landmarks. Is your algorithm always guaranteed to succeed? This is the parameterised verification problem. It is parameterised because you do not want to test your algorithm only on this particular part of Manhattan, but on all, potentially infinitely many, such environments.

We believe that there are many scenarios of multi-agent systems where navigating using exact positioning is not possible or desirable. On the one hand, precise positioning requires complicated hardware (such as GPS receivers, servo

¹Even if one has a map which is to scale, one probably does not navigate by measuring or calculating the distance one travels.

mechanisms, etc.), and on the other hand, even if the agent knows its exact location, many times it does not have this information for other objects in the environment, e.g., if a complete to-scale map of the environment is not available, and only a partial schematic exists (such as a “map” drawn on a napkin), or if there is no map at all.

Aims and Contributions. In order to capture the motivating example above, as well as many other scenarios, the aim of this work is to provide a formal framework for reasoning mathematically and computationally about multiple mobile-agents in grid-environments, having some tasks to perform. We model agents as finite-state machines that walk on grids containing obstructions. The agents can move along the axes, e.g., in two dimensions they can move horizontally and vertically but not diagonally. Agents maintain a fixed number of beacons which they can place on the grid to remember positions they have been to, and can be used for navigation or coordination with other agents. Grid-environments are parameterised by their size and the relative positions of the obstacles on the grid. Tasks are formalised in a suitable linear-time logic that extends LTL by adding the ability to talk about the relative positions of the agents and obstacles at different points in time. In particular, it can express tasks like gathering, border patrol, and line-formation. Agents move asynchronously according to an optional scheduling policy; this is motivated by the assumption that agents have no common notion of time since they have different internal clocks [40, 2, 6].

We prove that the verification problem is decidable (and in PSPACE) under two restrictions: i) there is a bounded number of obstacles, and ii) the agents are not allowed to issue commands like “increase my x -coordinate by 1” but can only issue commands that change their relative positions, e.g., “increase my x -coordinate until I go past this wall”. We prove PSPACE-hardness already for the verification problem of a single agent on singleton parameterised environments with no obstacles and no beacons. We therefore find it remarkable that the PSPACE-upper bound also holds for the verification problem with multiple agents, beacons, parameterised environments and multiple obstacles.

In stark contrast, we prove that verification is undecidable, even for very simple tasks (such as reachability), if either of these two conditions is relaxed. E.g., even on empty grids, thus i) holds, if agents can issue commands with absolute distances then verification is undecidable; and, if ii) holds, but there is no bound on the number of obstacles, then verification is undecidable even on simple “Avenues-and-Street” grids.²

The importance of this work is that it is the first to give a sound and complete decision procedure for the verification problem on parameterised grid-like environments. Previous work either involved abstractions [37], only a single environment [36], parameterised environments but with scheduling that restricted the number of turns [2], or parameterised environments but excluded grids altogether [45]. Thus, completely new ideas are needed.

Our work is based on the insight that the ability to move precisely in a coordinate system results in undecidability, and that methods from formal methods can shed light on

²Moreover, we get undecidability even if the absolute placement of the obstacles is not fixed, e.g., the width of the avenues may vary.

weakened agents that still retain the ability of moving relative to other objects. In this way we obtain a powerful new paradigm for modelling multi agent systems.

Related work. The parameterised model-checking problem has been considered in the verification community for models of (typically immobile) processes — see [6] for a survey. On the other hand, the distributed-computing literature consideration of mobile robots (i.e., automata walking on graphs) is mostly mathematical rather than algorithmic, but naturally (and interestingly) provides theorems that are parameterised (in, e.g., the number of robots, the number of internal states of the robots, etc.) [8, 38, 28, 20, 15, 14, 27].

The main tool we exploit for decidability is a spatio temporal logic called constraint LTL that was originally defined for the verification of counter machines [17, 18].

Our paper considers qualitative tasks. In contrast, [43] studies the shortest path problem in a 2D model that bears some similarity to ours: obstacles are non-intersecting rectangles with sides parallel to the axes, and obstacles that are in a straight line of sight can be seen, and also measured.

Our beacons are reminiscent of pebble automata [21, 4] which themselves are related to extensions of first-order logic with transitive closure; we remark that in our model there is no stack-like restriction on the orders that beacons can placed or retrieved, and yet we still achieve decidability.

Our paper considers verification. A related problem is synthesis with partially known environments, also known as generalised planning, which typically considers one dimensional environments [16, 32, 34, 35].

Besides navigation in known environments, the AAMAS community has studied distributed solutions to navigation tasks in unknown environments, and in contrast to our work, most studies were experimentally verified: [33] contains a technique for computationally sophisticated robots to solve the multi-robot coverage task problem using Voronoi partitioning; In [12], computationally and memory limited agents are studied using coalition logic to manage team-formation; in [29], the agents communicate by exchanging messages with a predefined set of other agents, whose task is to reach a goal state in weighted planar graphs where robots discover adjacent nodes to visited nodes; [24] contains a variation of the A^* algorithm for finding optimal paths in partially known graphs, i.e., where robots discover nodes adjacent to visited ones; and the authors of [11], motivated by reducing deployment time, consider intelligent cameras and autonomous robots for a museum guiding service.

2. NOTATION AND BACKGROUND

Notation. Let \mathbb{N} denote the integers, and \mathbb{N}_0 be $\mathbb{N} \cup \{0\}$. For $N \in \mathbb{N}$, write $[N]$ for the set $\{1, 2, \dots, N\}$. Fix an infinite set var of variables (that vary over \mathbb{N}). Fix, once and for all, a dimension $D \in \mathbb{N}$ for the grids (e.g., if $D = 1$ then the agents move on a line, if $D = 2$ then they move on planar grids, etc.). Write ϵ_i for the D -tuple $(0, \dots, 0, 1, 0, \dots, 0)$ that is 0 everywhere except for the i th co-ordinate which is 1. Write $\bar{0}$ for the D -tuple $(0, \dots, 0)$. Write $\bar{x}[i \leftarrow a]$ for the tuple \bar{x} with the value of the i th co-ordinate replaced by the a , e.g., $(0, 0, 1)[2 \leftarrow 1]$ is $(0, 1, 1)$. Throughout, we freely interchange between functional notation $f(i) = b$ and vector notation (\dots, b, \dots) . For an infinite sequence σ , write σ_i for the i th element of σ .

Two-counter Machines. Our undecidability proofs proceed by reducing the halting problem of two counter machines to the verification problem. An *input-free 2-counter machine* (2CM) [42] is a deterministic program manipulating two nonnegative integer counters using commands that can increment a counter by 1, decrement a counter by 1, and check whether a counter is equal to zero. We refer to the “line numbers” of the program code as the “states” of the machine. One of these states is called the *halting state*, and once it is entered the machine *halts*. The *non-halting problem for 2CMs*, which is known to be undecidable [42], is to decide given a 2CM M whether it does not halt.

Constraint linear-temporal logic (C-LTL). Linear temporal logic (LTL) is nowadays a well-established logic for reasoning about reactive systems. We recall the definition of constraint linear-temporal logic (C-LTL) introduced in [18]. C-LTL is a spatio-temporal logic that has been introduced for the verification of constraint automata, a class of automata that can be viewed as abstractions of counter-automata. It is the main tool that allows us to achieve decidability of the verification problem for multi-agent systems on grids.

A *term* t is either an element of \mathbb{N}_0 or an expression of the form $O^i(x)$ ($x \in \text{var}$ and $i \in \mathbb{N}_0$). We identify $O^0(x)$ with x . An *atomic constraint* R is an expression of the form $t \bowtie t'$ where t, t' are terms and \bowtie is one of $=, <, \equiv_a^b$ ($0 \leq a < b \in \mathbb{N}_0$). Here $=, <$ are interpreted as usual on \mathbb{N}_0 , and \equiv_a^b is the modulo unary relation, i.e., $\equiv_a^b(x)$ iff $x = a \bmod b$. E.g., $O^2(x) < O^1(y)$ is an atomic constraint. Informally, it means that the value of x two time steps ahead is less than the value of y one time step ahead. A *constraint* is a Boolean combination of atomic constraints. A *number constraint* is a constraint that only involves elements of \mathbb{N}_0 and variables, i.e., no terms of the form $O^i(x)$ for $i > 0$.

The syntax of C-LTL is given by: $\varphi ::= R \mid \neg\varphi \mid (\varphi \vee \varphi) \mid O\varphi \mid \varphi U \varphi$ where R is an atomic constraint. A *valuation* is a function $\text{var} \rightarrow \mathbb{N}_0$. Models of C-LTL formulas are infinite sequences of valuations, i.e., $\sigma = \sigma_0 \sigma_1 \dots$ where each σ_j is a valuation. The semantics of C-LTL follows:

$$\begin{aligned} (\sigma, i) \models R &\quad \text{iff } \sigma_{i+l}(x) \bowtie \sigma_{i+m}(y) \text{ and} \\ &\quad R = O^l(x) \bowtie O^m(y) \\ (\sigma, i) \models \neg\varphi &\quad \text{iff } (\sigma, i) \not\models \varphi \\ (\sigma, i) \models \varphi \vee \psi &\quad \text{iff } (\sigma, i) \models \varphi \text{ or } (\sigma, i) \models \psi \\ (\sigma, i) \models O\varphi &\quad \text{iff } (\sigma, i+1) \models \varphi \\ (\sigma, i) \models \varphi U \psi &\quad \text{iff } \exists j \geq i : (\sigma, j) \models \psi \text{ and} \\ &\quad \forall l \in [i, j) : (\sigma, l) \models \varphi \end{aligned}$$

Write $\sigma \models \varphi$ if $(\sigma, 0) \models \varphi$. We use the usual derived operators $F\varphi := \text{true} U \varphi$, its dual $G\varphi := \neg F \neg\varphi$, and $\varphi_1 R \varphi_2$ which is the dual of U , i.e., $\neg(\neg\varphi_1 U \neg\varphi_2)$, as well as the short-hands \wedge and \vee . For example, $G(x < O^1(x))$ means that “globally the value of x at the current point of time is smaller than the value of x at the next point of time”, which simply means that x is always strictly increasing.

We mention the following important facts about C-LTL, see [18]. First, recall that linear-temporal logic (LTL) has the same syntax as C-LTL except that instead of atomic constraints it has atomic propositions such as p and q ; and LTL formulas are interpreted over infinite sequences of sets of atomic propositions. Now, C-LTL generalizes LTL (the idea is to associate to each atom p a unique variable x , and

replace p by the constraint $x = 1$ and replace $\neg p$ by the constraint $x = 0$). Second, the satisfiability problem (and thus also the validity problem) for C-LTL (i.e., given a C-LTL formula φ , decide if there exists σ such that $\sigma \models \varphi$) is decidable, and in fact has the same complexity as for LTL, i.e., it is PSPACE-complete.

3. ENVIRONMENTS AND AGENTS

Environments. We model environments as grids of arbitrary, but fixed, dimension containing obstacles whose faces are parallel to the axes. A (finite or infinite) set of environments of interest is called a *parameterised environment*. Comparisons between points of \mathbb{N}^D are taken point-wise, e.g., $\bar{v} \leq \bar{w}$ means that $v_i \leq w_i$ for $i \leq d$. We describe various subsets of \mathbb{N}^D . A *rectangle* is a subset of \mathbb{N}^D of the form $\{\bar{a} \in \mathbb{N}^D : \bar{v} \leq \bar{a} \leq \bar{w}\}$, where $\bar{v} \leq \bar{w} \in \mathbb{N}^D$. The points \bar{v} and \bar{w} are called the *SW-* and *NE-corners* of the rectangle, respectively.³ The *interior* of a rectangle R , whose SW and NE corners are \bar{v} and \bar{w} , is the set $R^{int} := \{\bar{a} \in \mathbb{N}^D : \bar{v} < \bar{a} < \bar{w}\}$.

A *K-obstruction* $M = \langle M_1, \dots, M_k \rangle$ is a tuple of K many rectangles. Given $L \in \mathbb{N}$, and a *K-obstruction* M such that all rectangles in M are contained in $[L]^D$, the (L, M) -environment is the graph $G = (V, E)$ whose vertex set $V = [L]^D \setminus (\bigcup_i M^{int})$ (i.e., it is the rectangle with corners $(0, \dots, 0)$ and (L, \dots, L) , and the interiors of the rectangles in M removed), and whose edge relation $E \subset V \times V$ consists of pairs (\bar{v}, \bar{w}) such that for some i we have $|v_i - w_i| = 1$ and for $j \neq i$ we have $v_i = w_i$ (i.e., horizontally- or vertically-adjacent vertices are joined with an edge). Environments are also called finite labyrinths [30]. A *parameterised environment* is some finite or infinite set of environments.

Since only the interiors of the obstructing rectangles are removed, agents are allowed to move along the “walls” of an obstacle. Observe that this has the effect that if, for example, two rectangles (or a rectangle and a boundary of the grid) have a common edge/face, agents can still walk between them “along the adjoining wall”. If one wants to ensure, that agents cannot walk between two rectangles, one can specify that the rectangles overlap. We add that overlapping rectangles can be used to build obstacles of different shapes than rectangles. For example, one can build a non-rectangular obstacle such as an “L” by using two rectangles and specifying that they have the same SW-corner, and that one rectangle is thinner and higher and the other rectangle is broader and more shallow. We finally add, that if one wants to remove the points common to a rectangle and a boundary of the grid⁴ this could be accomplished by adding a second type of rectangle with the desired behaviour. For ease of exposition we refrain from doing so.

Agents. Agents are modeled as finite-state machines that can move along the axes of the grid. E.g., in a 2-dimensional grid, each single move is in direction north, south, east or west, but not diagonally. More formally, an agent A is tuple (Q, I, δ) where Q is a finite set of *states* (we usually assume

³Note that even though our grids are not necessarily 2-dimensional this is perhaps the most common case, and we find it assists in imagining things if we use 2-dimensional terms like “NE”.

⁴This can be used also to simulate a grid with sides of different length by positioning suitable obstacles along the boundaries.

w.l.o.g. that $Q = [|Q|]$; $I \subseteq Q$ is a set of *initial states*; $\delta \subseteq Q \times GC \times Q$ is a finite *transition relation*. Elements of GC are called *guarded commands* and are of the form $(d, guard)$ where $d \in [D]$ is the axis along which the agent should move, and *guard* is a condition specifying some restriction that the path taken by the agent during this move must satisfy for the move to be possible. For example, the guard may specify that the agent moves exactly one step in the positive direction of the axis. There are many possible definitions for the types of guards that can be used, and these choices determine the “power” the agents have, and we will consider a few possible definitions of GC in later sections. In all cases, we assume that the guard can test whether the agent is at a boundary of the grid. Observe that we implicitly assume⁵ that i) the path taken by an agent in a single command does not collide with any other agent (in more detail, an agent may start and end in the same position as another agent, but it may not “pass-through” an agent in a single move); and ii) the path taken does not use positions occupied by an obstruction. Note that the guard may be such that it does not fully specify the end position of a move, in which case the system nondeterministically chooses it (in a way which satisfies the guard).

4. UNDECIDABILITY RESULTS

In this section we give two simple undecidability results that direct our choice of model for agents in parameterized grid environments, presented in the next section.

A common assumption is that agents have the ability to exactly know and/or control their position [41, 27, 38, 39, 36]. This assumption abstracts real-world situations of agents that, for example, are equipped with a global positioning system (GPS). Unfortunately, as demonstrated by the following theorem, it is very easy to show that making this assumption leads to undecidability of the verification problem with respect to even the simplest tasks and parameterized grid environments. It is worth noting that this assumption can be made, without leading to undecidability, in many cases where the environments of interest are not grids, e.g., environments with bounded tree-width [45, 2].

Say that an agent has *precise positioning* if GC contains (directly or indirectly) commands of the form $(d, x_d = x'_d + j)$, with $j \in \{0, -1, 1\}$, where x_d, x'_d are the positions of the agent along the d axis at the start and end of the move, and guards that allow the agent to test its position, in particular whether it is on the bottom or left boundary of the grid.

THEOREM 1. *The following problem is undecidable: given a single agent with precise positioning, and a state h of the agent, verify that for all obstacle-free finite 2-dimensional grids no run of the agent ever enters state h .*

PROOF. The proof is by reduction from the non-halting problem of 2-counter machines. Given a 2CM M , we build an agent A that has the same states as the 2CM plus an additional *overflow* state. The agent simulates the 2CM by using its position (x, y) to encode a configuration of M where the first counter has value x and the second counter has value y . The agent begins by going to the origin of the grid (south as far as possible then west as far as possible). Incrementing (resp. decrementing) a counter is done by simply moving one

⁵In other words, an instruction can be taken only if these extra conditions are satisfied.

step in the correct direction. A test for zero amounts to the agent checking that it has collided with the bottom or left boundary of the grid. If the agent wants to go right or up (to simulate an increment), but hits the boundary, it enters the special *overflow* state and discontinues the simulation.

It is easy to see that, on an $L \times L$ grid, the agent can simulate any prefix of the computation of the 2CM in which the values of both counters never go above n . Thus, M does not halt iff, for all $L \in \mathbb{N}$ the agent A doesn’t enter the halting state in the $L \times L$ grid. \square

Observe that it is simple to modify Theorem 1 and obtain that, if one wishes to consider grids of unbounded size, verifying any non-trivial task for an agent with precise positioning is undecidable. Hence, if we wish to be able to automatically verify that the agent performs its task on a families of grids of unbounded size we must limit the agent’s ability to precisely control/know its location.

In our quest for a model for describing agents in parameterized grid environments we are motivated by the example from the introduction of a person navigating in an unknown city using a schematic map, or instructions from a GPS unit. Note that one is usually *not* using the precise positioning information available to the GPS, but one is simply following instructions such as “turn left at the next corner”. All that is needed to follow such an instruction is the ability to detect the next corner. Let us consider then a model for the agent where the guarded command can specify a guard that expresses (directly or indirectly) that the end position \bar{x}' of the move is a vertex of some obstacle in the environment, and no point strictly between the start point and the end point is a vertex of an obstacle.⁶ This is arguably a very basic way of detecting a corner, as one is doing it “by feel” (reminiscent of how a blind person uses a white cane). We call agents with this ability agents with *next corner detection*.

For the following theorem, given $L \in \mathbb{N}$, let the L -regular-city be the 2-dimensional grid environment of length $2L+1$, in which for every $i, j \in \{0, \dots, L\}$ there is a 1×1 obstacle whose SW corner is located at $(2i, 2j)$.

THEOREM 2. *The following problem is undecidable: given a single agent with next corner detection, and a state h of the agent, verify that for all $L \in \mathbb{N}$, no run of the agent ever enters state h while navigating the L -regular-city.*

PROOF. The proof is very similar to that of Theorem 1. Unlike Theorem 1, we do not assume that the agent has precise positioning, and thus it can not encode the counters of the 2CM directly by its position. However, it can do so indirectly as follows: it encodes the value of the first counter by the number of obstacles that are strictly to the west of it (at the same y coordinate as it is), and the value of the second counter by the number of obstacles that are strictly to the south of it (at the same x coordinate). The main difference with the agent in the proof of Theorem 1, is that incrementing (resp. decrementing) a counter is done by moving to the SW corner of the next obstacle in the correct direction, which takes two moves. For example, if q is an increment of the first counter, the agent moves twice to the next corner to the east: the first move takes it from the SW corner of the current obstacle to its SE corner (and to state

⁶If one wishes, one can further assume that if no such \bar{x}' exists then the agent walks as far as it can.

q'), and the next move takes it to the SW corner of the next obstacle (and the next state of the 2CM).

It follows that, in an L -regular-city, the agent can simulate any prefix of the computation of M in which the counter values don't exceed L . Thus, M does not halt iff, for all $L \in \mathbb{N}$, the agent never enters the halting state of M . \square

As was the case with Theorem 1, one can modify the proof above to obtain undecidability for any non-trivial task. The proof goes through unchanged even if the definition of an L -regular-city is modified to allow the obstacles and the gaps between them to have different sizes, as long as the general structure of aligned streets and avenues is maintained.

5. A MODEL OF MAS ON GRIDS

In this section we present a model of multiple agents operating asynchronously in unbounded parameterized grid environments with a bounded number of obstacles. We also define a way to specify the tasks for these agents.

The theorems in the previous section imply that if we want to be able to decide whether our agents achieve their specified tasks on grid environments of unbounded size, and still maintain the agents' ability to detect corners of obstacles, we should disallow the agents from precisely specifying the distance they travel as well as bound the number of obstacles in a parameterized environment. However, we place no bounds on the sizes and positions of the obstacles.

Informally, the guarded commands of the agents do not allow them to specify absolute positions or step sizes, and only allow one to express relative positioning with respect to other objects in the environment. In order to regain some of the power lost by this relative positioning, agents are also allowed to place some *fixed* number of smart markers, called *beacons* (similar to non-directional radio beacons used in air and marine navigation), in the environment. At any point in time, and from any point in the environment, an agent can test whether its position along any of the D axes is smaller, equal, or larger than that of any other agent, corner of an obstacle, or any beacon (of any agent). A beacon can also be remotely retrieved by the agent that owns it (i.e., without going back to it) and deposited at the current location of the agent. Agents are also able to query other agents as to their current local state.

Note that the beacons allow the agents to perform certain operations that would otherwise be impossible with relative positioning, e.g., such as marking a position an agent can later return precisely to, or to draw a virtual line in the town square such that the agents later gather in the square with half of them on each side of the line. The beacons can be implemented, for example, by making the following (limited) use of a GPS system and memory registers: placing a beacon number i of agent j in the current position is *simulated* by j recording, in its i 'th register, the current position as indicated by the GPS. Comparing an agent's position to that of the beacon is done by querying the GPS system as to whether some coordinate d of the current location is smaller, larger, or equal, to the value in the corresponding register. Retrieving the beacon is done by simply overwriting the value in the register. Obviously, not all the power of this model may be needed or possible to implement by a real system. For example, our agents can see through walls (I.e., they are in a "smart city" where the corners of each obstacle broadcast radio signals), and one is welcome to not

make use of any unnecessary feature (E.g., limit the agents to not see through obstacles).

For the rest of the paper fix the number B of beacons, the number N of agents, and the number K of obstacles.

Variables. We define the following variables (i.e., elements of *var*), that will be used later to define other important concepts such as guards, runs, etc.:

- *agent variables* *avar* := $\{x_{n,d} : n \in [N], d \in [D]\}$;
- *primed agent variables* *avar'* := $\{x'_{n,d} : n \in [N], d \in [D]\}$;
- *beacon variables* *bvar* := $\{b_{n,d}^j : n \in [N], d \in [D], j \in [B]\}$;
- *primed beacon variables* *bvar'* := $\{b'^j_{n,d} : n \in [N], d \in [D], j \in [B]\}$;
- *agent local-state variables* *svar* := $\{s_n : n \in [N]\}$;
- *obstruction variables* *ovar* := $\{u_{k,d}, v_{k,d} : k \in [K], d \in [D]\}$;
- the *size variable* l , and the *turn variable* *turn*.

The values of each variable have the following meanings. $\bar{x}_n := (x_{n,1}, \dots, x_{n,d})$, is the current position of agent n , and the primed version \bar{x}'_n is the next position of agent n ; the position of the j 'th beacon of agent n is $\bar{b}_n^j := (b_{n,1}^j, \dots, b_{n,d}^j)$, and its next position is $\bar{b}'_n^j := (b'^j_{n,1}, \dots, b'^j_{n,d})$. The value of s_n is the current local state of the n th agent. The variables \bar{u}_k and \bar{v}_k are the SW- and NE-corners of the k th obstacle; l is the length of the grid; and *turn* is which agent's turn it is (used to define schedules). For simplicity, we assume that the values of l and variables in *ovar* do not change over time, i.e., that the size of the grid and the positions of obstacles are fixed throughout a run of a system.

Parameterised environments with a fixed number of obstacles. An *environment constraint* is a number constraint ϕ over variables *ovar* $\cup \{l\}$. It determines a set of environments \mathcal{G}_ϕ as follows. Every (L, M) -environment G with $M = \langle M_1, \dots, M_K \rangle$ determines a valuation σ_G over *ovar* $\cup \{l\}$ as follows: σ_G maps variable l to L , and for each $i \in [K]$, σ_G maps \bar{u}_i (resp. \bar{v}_i) to the SW-corners (resp. NE-corner) of the rectangle M_i . Thus, a number constraint ϕ determines the set \mathcal{G}_ϕ of environments G such that the valuation σ_G satisfies ϕ . E.g., for $d = 2, K = 1$, and 2-tuples of variables $(u_1, u_2), (v_1, v_2)$ representing the inner rectangle, the constraint $l \geq 10 \wedge \bigwedge_{i=1,2} 0 < u_i < v_i < l$ determines the set of rectangular "race-tracks" of size at least 10.

Note that by constraining the variable l , we may specify environments of a single size (e.g., $l = 4$), a finite set of sizes (e.g., $l = 4 \vee l = 5$ or $l < 10$), or infinitely many sizes (e.g., $l > 10$ or $\equiv_0^2(l)$).

Agents. Recall from Section 3 that an agent A is tuple (Q, I, δ) where Q is a finite set of states, $I \subseteq Q$ is a set of initial states, and $\delta \subseteq Q \times GC \times Q$ is a finite transition relation, where the guarded commands GC are of the form $(d, guard)$ where $d \in [D]$. It remains to define the possible values of *guard*. We let *guard* be any number constraint over *avar* \cup *avar'* \cup *bvar* \cup *bvar'* \cup *ovar* $\cup \{l\}$. Note that a transition specifies that beacon j of agent n is retrieved and dropped in the new location of the agent by specifying in the

guard that $\bar{x}'_n = \bar{b}'^j_n$. An *agent ensemble* is a tuple of agents $\mathcal{A} = \langle A_1, \dots, A_N \rangle$ (such that all the agents' commands use the same N, K and B).

Configurations, Schedules, Runs. Let $G = (V, E)$ be an (L, M) -environment with K obstacles, and let \mathcal{A} be an agent ensemble. A *configuration* of \mathcal{A} on G is a tuple $c := (loc, locb, state)$, where $loc : [N] \rightarrow V$ maps an agent to its location; $locb : [N] \times [B] \rightarrow V$ maps a beacon to its location; and $state : [N] \rightarrow \cup_j Q_j$, such that $state(j) \in Q_j$ (for $j \in [N]$), maps an agent to its current local state. Say that c is an *initial configuration* if $state(j) \in I_j$ (for $j \in [N]$).

We can think of c as a valuation val_c over variables $avar \cup bvar \cup ovar \cup svar$ that (for $n \in [N], j \in [B], k \in [K]$): maps \bar{x}_n to $loc(n)$, maps \bar{b}_n^j to $locb(n, j)$, maps \bar{u}_k (resp. \bar{v}_k) to the SW-corner (resp. NE-corner) of the k th obstruction M_k of the environment G ; and maps s_n to $state(n)$. Similarly, if we are given a second configuration c' , we can further extend val_c to get a valuation $val_{c,c'}$ that maps \bar{x}'_n to $loc'(n)$ and \bar{b}'^j_n to $locb'(n, j)$.

For configurations $c = (loc, locb, state)$ and $c' = (loc', locb', state')$, and $n \in [N]$, write $c \vdash_n c'$ if agent n can, by taking one transition, change the configuration from c to c' . Formally, $c \vdash_n c'$ if there exists $(q, (d, guard), q') \in \delta_n$ and $\alpha \in \mathbb{Z}$ such that:

1. $state(n) = q$, and $state' = state[n \leftarrow q']$ (i.e., the agent changes state from q to q');
2. $loc' = loc[n \leftarrow loc(n) + \alpha e_d]$ (i.e., the agent moves some distance $\alpha \in \mathbb{Z}$ along the d th axis);
3. For every $m \in [N], j \in B$, either $loc'(m, j) = locb(m, j)$, or $m = n$ and $locb'(m, j) = loc'(n)$; (i.e., the agent can transfer any of its beacons to its new location);
4. $\sigma_{loc, loc'} \models guard$ (i.e., the guard holds);
5. for every $\bar{v} \in \mathbb{N}_0^D$ strictly between $loc(n)$ and $loc'(n)$: $\bar{v} \in V$ (i.e., no obstruction) and $\bar{v} \neq loc(j)$ for all $j \in [N] \setminus \{n\}$ (i.e., no collision).

An *schedule* κ is an element of $[N]^\omega$. A set S of schedulers is called a *scheduler*. A *scheduler constraint* is a C-LTL formula ω over the variable $\{turn\}$. It induces the set of schedules κ such that there exists $\sigma \models \omega$ such that $\sigma_i \models (turn = \kappa_i)$ for all $i \in \mathbb{N}$.⁷

EXAMPLE 1. Round-robin of N agents may be expressed as the N -scheduler consisting of the schedules $(\pi(1) \dots \pi(N))^\omega$ such that π is a bijection of $[N]$. To express this scheduler in C-LTL one may use the formula $\bigvee_\pi \mathsf{G} \bigwedge_n (turn = n) \rightarrow \mathsf{O}(turn = \pi(\pi^{-1}(n) + 1))$ where the disjunction is over all bijections π of $[N]$.

EXAMPLE 2. Fair scheduling of N agents may be expressed as the set of schedules satisfying $\bigwedge_n \mathsf{GF}(turn = n)$.

A *run* ρ of \mathcal{A} on G according to scheduler S is an infinite sequence $\rho = \rho_1 \rho_2 \rho_3 \dots$ of configurations such that ρ_1 is initial and there exists $\kappa \in S$ such that $\rho_i \vdash_{\kappa_i} \rho_{i+1}$, for all i . The *agent locations* of the run is the sequence

⁷It is also possible to define more powerful schedulers by C-LTL formulae over the variables $\{turn\} \cup svar \cup avar \cup bvar \cup ovar \cup \{l\}$. For ease of exposition we refrain from doing so.

$loc(\rho_1)loc(\rho_2) \dots$, the *beacon locations of the run* is the sequence $locb(\rho_1)locb(\rho_2) \dots$, and the *states of the run* is the sequence $state(\rho_1)state(\rho_2) \dots$. To every run $\rho = \rho_1 \rho_2 \dots$ we associate $val(\rho)$, which is the sequence of valuations $val_{\rho_1} val_{\rho_2} \dots$.

Agent Tasks. Agents should achieve some task in their environment. A *task* is a C-LTL formula τ over variables $avar \cup svar \cup bvar \cup ovar \cup \{l\}$. The ensemble \mathcal{A} *achieve the task* τ in environment G according to scheduler S if for all runs ρ of \mathcal{A} on G according to scheduler S , the sequence of valuations $val(\rho)$, satisfies τ . Note that the task may, if one wishes, restrict the initial states and positions of the agents. For example, agents may be required to start at the corners of the grid, etc.

EXAMPLE 3. Agents gather if eventually they arrive at the same, not previously determined, location [39]. This task can be expressed by the C-LTL formula $\mathsf{F} \bigwedge \{x_{n,d} = x_{m,d} : n, m \in [N], d \in [D]\}$.

EXAMPLE 4. The Line Formation task requires the set of agents to form a line, an example of a pattern-formation task [25]. This can be expressed by the C-LTL formula: $\mathsf{F} \text{Line}$ where $\text{Line} := \bigvee_{d \in [D]} \bigwedge \{x_{n,d} = x_{m,d} : n, m \in [N]\}$. A variation asks that the agents repeatedly form a line, and can be written $\mathsf{GF} \text{Lin}$.

The following is inspired by the task of a guard making sure that the doors of all buildings on campus are locked.

Example.

We consider the task of **border patrol** for a **single agent**. This task consists of moving through the grid such that every border, in our encoding *every edge of every obstacle*, is traversed at least once by the agent. Below we will give a specification for border patrol and describe a protocol for an agent that achieves border patrol. Our main positive result (Theorem 3) means that one can automatically verify this protocol against this specification for every parameterized grid environment satisfying the constraints of Theorem 3. For ease of exposition, we consider the 2-D setting where all obstacles are disjoint (disjointness can be encoded by a C-LTL-formula which is quadratic in the number of obstacles).

Specification for border patrol. We consider an obstacle described by SW corner (x_1, y_1) and NE (x_2, y_2) . For the edge between (x_1, y_1) and (x_2, y_2) (the bottom edge of the rectangle) we can encode by an *eventually formula* the agent arrives at the corner (x_1, y_1) or (x_1, y_2) . Then we can describe by an *until formula* that the agent will eventually reach the other corner of the edge while not leaving the edge on the way. Border control is specified by a conjunction of formulae as described above for every edge of every obstacle.

Protocol for border control. We assume that the agent starts at the origin (where the security personnel office is located), i.e., the SW corner of the grid. When the agent is at the left side of the grid it places a beacon on its position. Then the agent moves from the left of the grid to the right of the grid circling around every obstacle encountered on its way. After circling around an obstacle the agent uses the beacon to stay on the horizontal line defined by the coordinates of the beacon. When the agent has arrived at the right hand side of the grid it returns to the beacon. Then the agent moves upwards until it is on the same height with

the next obstacle, i.e., the agent and the SW corner of the first obstacle in the whole environment whose SW corner is at a larger y coordinate than its current position. The agent repeats this behavior until it reached the NE corner of the grid. This completes the description of the agent. It is interesting to note that the described agent can be implemented by a finite automaton with a single beacon whose size only depends linearly on the number of obstacles.

Observe that in the description above, the agent's move up until aligned with the SW corner of the next obstacle cannot be directly implemented in a model where the agent can only use vision to navigate (since that corner may be obstructed by other obstacles that are horizontally between the agent and that corner), and the example illustrates the power of our model. As noted before, one can obviously limit the agents to not use the full power of the model, and one can come up with border patrol protocols for vision/touch limited agents as well.

6. PARAMETERISED VERIFICATION

We first formalise the parameterised verification problem (PVP) for agents on parameterized grids with a bounded number of obstacles (as defined in the previous section) and show that it is decidable. The PVP depends on formulas for tasks, environments, and schedulers. Recall that \mathcal{G}_ϕ is the (possibly infinite) set of environments determined by ϕ , and S_ω is the set of schedules determined by ω .

DEFINITION 1. *The parameterised verification problem of Sliding Robots on Grids, written PVP, is the following decision problem: given an agent ensemble \mathcal{A} , a task τ for the agents, an environment constraint ϕ describing a parameterized grid environment, and scheduler constraint ω , decide whether for every $G \in \mathcal{G}_\phi$, the agents \mathcal{A} achieve task τ on environment G according to scheduler S_ω .*

THEOREM 3. *The PVP is in PSPACE.*

PROOF. We show this by reducing the PVP to satisfiability of C-LTL which is in PSPACE [18]. I.e., we effectively transform the agent-ensemble $\mathcal{A} = \langle A_1, \dots, A_N \rangle$, environment constraint ϕ , and scheduler constraint ω , into a C-LTL formula $\psi_{\mathcal{A}, \phi, \omega}$ whose models code all, and only, the runs of \mathcal{A} on environments $G \in \mathcal{G}_\phi$ according to the scheduler S_ω . Then we check that the C-LTL formula $\psi_{\mathcal{A}, \phi, \omega} \rightarrow \tau$ is valid, or equivalently, that $\neg(\psi_{\mathcal{A}, \phi, \omega} \rightarrow \tau)$ is not satisfiable.

Here are the details of the transformation. Say $A_n = (Q_n, I_n, \delta_n)$, and w.l.o.g. assume, for $n \in [N], j \in Q_n$, that $Q_n = [[Q_n]]$. The formula $\psi_{\mathcal{A}, \phi}$ has variables $\text{avar} \cup \text{ovar} \cup \{l\} \cup \{s_i : i \in [N]\} \cup \{\text{turn}\}$ and uses constants from the set $\{0, 1, \dots, \max_n |Q_n|\}$. We first define some helper formulas:

- $In(\bar{x}, \bar{u}, \bar{v})$ is $\bigwedge_d u_d \leq x_d \leq v_d$ (position \bar{x} is within the rectangle determined by \bar{u} and \bar{v});
- NotObstructed is $\bigwedge_n \bigwedge_k \neg In(\bar{x}_n, \bar{u}_k, \bar{v}_k)$ (no agent is inside any rectangle obstruction);
- $Btwn_d(\bar{x}, \bar{y}, \bar{z})$ is $(x_d < y_d < z_d \vee z_d < y_d < x_d) \wedge \bigwedge_{e \neq d} z_e = y_e = x_e$ (position \bar{y} is between positions \bar{x} and \bar{z} and lie parallel to axis d);
- $O^1(\bar{z})$ is $(O^1(z_1), \dots, O^1(z_n))$.

Define $\psi_{\mathcal{A}, \phi, \omega}$ as the conjunction of:

- $G \text{ NotObstructed}$ (agents are unobstructed);
- $\phi \wedge G[l = O^1(l) \wedge \bigwedge_k \bar{u}_k = O^1(\bar{u}_k) \wedge \bar{v}_k = O^1(\bar{v}_k)]$ (obstructions satisfy the environment constraint and do not change over time);
- $\bigwedge_n \bigvee_{j \in I_n} s_n = j$ (each agent starts in an initial state);
- $\omega \wedge G \bigwedge_n (\text{turn} \neq n) \rightarrow (\bar{x}_n = O^1(\bar{x}_n) \wedge s_n = O^1(s_n))$ (agents take turns according to a schedule in ω);
- $G \bigwedge_n \bigwedge_{i \in B} (\text{turn} \neq n) \rightarrow (\bar{b}_n^i = O^1(\bar{b}_n^i))$ (only beacons of the agent whose turn it is can be moved);
- $\bigwedge_{n,j} G[\text{turn} = n \wedge s_n = j \rightarrow \bigvee_{t \in \delta_n} Next_t]$ (agents follow their protocols),

where $Next_t$, for t of the form $(j, (d, \text{guard}), j')$, is

$$\text{guard}' \wedge move \wedge O^1(s_n) = j' \wedge \text{nocollide} \wedge \text{beacon}$$

where guard' is guard with every primed agent variable $x'_{m,e}$ (resp. beacon variable $b'_{m,e}$) replaced by $O^1(x_{m,e})$ (resp. $O^1(b_{m,e})$); $move$ is $\bigwedge_{e \neq d} x_{n,e} = O^1(x_{n,e})$; nocollide is $\bigwedge_{m \neq n} \neg Btwn_d(\bar{x}_n, \bar{x}_m, O^1(\bar{x}_n))$; and beacon is $\bigwedge_i \bar{b}_n^i \neq O^1(\bar{b}_n^i) \rightarrow O^1(\bar{b}_n^i) = O^1(\bar{x}_n)$ (the agent can move a beacon only to its new location).

It is not hard to check that for every sequence σ of valuations, $\sigma \models \psi_{\mathcal{A}, \phi, \omega}$ if and only if there exists $G \in \mathcal{G}_\phi$ and a run ρ of \mathcal{A} on G according to scheduler S_ω such that (i) $\sigma \upharpoonright \text{avar}$ is equal to $\text{loc}(\rho)$; (ii) $\sigma \upharpoonright \{s_n : n \in [N]\}$ is equal to $\text{state}(\rho)$; (iii) $\sigma \upharpoonright \text{ovar} \cup \{l\}$ is a sequence of identical valuations, each satisfying ϕ .

Thus, $\psi_{\mathcal{A}, \phi, \omega} \rightarrow \tau$ is not valid if and only if there exists a sequence σ satisfying $\psi_{\mathcal{A}, \phi, \omega} \wedge \neg\tau$ if and only if there exists $G \in \mathcal{G}_\phi$ and a run ρ of \mathcal{A} on G according to scheduler S_ω such that the sequence $\text{val}(\rho)$ of valuations of the run ρ satisfies $\neg\tau$ if and only if the agents \mathcal{A} do not achieve the task τ according to scheduler S_ω on all environments from \mathcal{G}_ϕ . This completes the proof. \square

Before presenting a PSPACE lower-bound for the PVP, we need some definitions and a lemma.

Let AP be a set of atomic propositions, and let $\text{AP}_{ig} := \text{AP} \cup \{\text{ignore}\}$, where ignore is a new atomic proposition not in AP . Let $\Sigma := 2^{\text{AP}}$ and $\Sigma_{ig} := 2^{\text{AP}_{ig}}$. Given a word $w' \in \Sigma_{ig}^\omega$, let $\text{sub}(w)$ be the word obtained by deleting all letters in w that are not in Σ (i.e., which contain ignore).

LEMMA 1. *Let ϕ be an LTL formula over atomic propositions AP . One can compute in polynomial time an LTL formula ϕ' over AP_{ig} , such that for every $w' \in \Sigma_{ig}^\omega$ we have that $w' \models \phi'$ iff $\text{sub}(w)$ is infinite and models ϕ .*

PROOF. W.l.o.g., we assume that ϕ is in positive normal form (i.e., negations are pushed to the atoms). The required formula is the conjunction of two formulas: the first requires that $\neg\text{ignore}$ holds infinitely often (thus ensuring that if $w' \models \phi$ then $\text{sub}(w)$ is infinite), and the second “simulates” ϕ on the points where $\neg\text{ignore}$ holds (and “skipping” the points in which ignore holds). Formally, $\phi' := \hat{\phi} \wedge \text{GF}(\neg\text{ignore})$, where $\hat{\phi}$ is constructed by induction on the structure of ϕ as follows:

- if $\phi := t$, where t is an atomic proposition or its negation, then $\hat{\phi} := \text{ignore} \cup (\neg\text{ignore} \wedge t)$;

- if $\phi := \phi_1 \bowtie \phi_2$, where $\bowtie \in \{\vee, \wedge\}$, then $\widehat{\phi} := \widehat{\phi_1} \bowtie \widehat{\phi_2}$;
- if $\phi := \mathbf{O} \phi_1$ then $\widehat{\phi} := \text{ignore } \mathbf{U}(\neg \text{ignore} \wedge \mathbf{O} \widehat{\phi_1})$;
- if $\phi := \phi_1 \mathbf{U} \phi_2$ then $\widehat{\phi} := (\text{ignore} \vee \widehat{\phi_1}) \mathbf{U} (\neg \text{ignore} \wedge \widehat{\phi_2})$;
- if $\phi := \phi_1 \mathbf{R} \phi_2$ then $\widehat{\phi} := (\neg \text{ignore} \wedge \widehat{\phi_1}) \mathbf{R} (\text{ignore} \vee \widehat{\phi_2})$.

This completes the proof. \square

THEOREM 4. *PVP is PSPACE-hard already for the case of a single agent on singleton parameterized environments with no obstacles and no beacons.*

PROOF. We reduce the problem of validity of LTL formulas, which is PSPACE-hard, to the PVP. Let ϕ be an LTL formula over the atomic propositions (w.l.o.g.) $\mathbf{AP} := \{1, \dots, D\}$. Consider the parameterized environment G containing the single D -dimensional grid with sides of length 1 (i.e., the discrete d -dimensional unit hypercube), with no obstacles. Note that every position on this grid is a vector $\bar{x} \in \{0, 1\}^D$. Consider a single agent A with two states \top, \perp . The transition relation allows the agent, from each state \top, \perp , to transition to either \top, \perp while moving in any direction (or staying in the same place).

A configuration c of A on G encodes a set $X \in \Sigma_{ig}$ of atomic propositions in \mathbf{AP}_{ig} as follows: $i \in [D]$ is in X iff the i 'th coordinate of the location of A is 1, and $\text{ignore} \in X$ iff the local state of A is \perp . Hence, by the definition of A , for every word $w' \in \Sigma_{ig}^\omega$ there is some run of A on G that encodes w' , and vice-versa. Also, using the same encoding (of atomic propositions of \mathbf{AP}_{ig} as constraints on the location/state of the agent) we can write, with a linear blowup, a task τ that is a C-LTL formula that encodes the LTL formula ϕ' we obtain from ϕ using Lemma 1.

By Lemma 1, ϕ is valid iff ϕ' is valid. By the reasoning above, the later is true iff the output of the PVP is true on the input: agent A , a formula describing the hypercube G , and the task τ (and the unrestricted scheduler true). \square

Observe that, in the proof above, the location of the robot encodes the values of all atomic propositions in parallel, which is the reason D dimensions are used. However, we can make do instead with a single dimension if we encode the values in serial. I.e., if we partition the positions (over time) of the robot into blocks of length D , and let the j position inside a block encode the value of the j 'th atomic proposition at the time corresponding to the block number. I.e., for $i \in \mathbb{N}_0$, and $1 \leq j \leq D$, the position of the robot at time $iD + j$ encodes the value of atomic proposition number i at time j . By suitably modifying ϕ' and $\widehat{\phi}$ to this new encoding we can deduce that the problem remains PSPACE-hard also in the interesting cases of 2 and 3-dimensional grids.

7. DISCUSSION

Parameterized verification of multi-agent systems is a hard problem, and the case of most interest, i.e., that of 2 and 3-dimensional grids, even more so. Indeed, in [2] it is shown that (for agents with precise positioning) while the problem is decidable for many parameterized environments it is undecidable even for agents working on a 1-D grid with only collision-detection abilities. In this work we gave one way to regain decidability. We have presented a model for describing multiple agents moving in parameterised grid environments of arbitrary size with a fixed number of obstacles

of arbitrary sizes. We have shown that relaxing the requirement that agents can not specify exact absolute positioning or step size, or the requirement of having a bounded number of obstacles, results in undecidability of the verification problem of very simple tasks of even a single agent. Our model of agents is very powerful (without becoming undecidable), and it generalises many other models (such as vision and touch based agents).

Many works in the past reason about environments by using all kinds of abstractions into graphs [9, 10, 44, 46]. Our approach avoids such abstractions (except for discretising space) by directly reasoning about the grid environments. We believe that our choice of using C-LTL as a logic for specifying both agents, parameterised environments, and tasks, allows one to encode things in a direct and natural way, and yields an elegant automatic verification algorithm. We draw an analog to the case of timed-automata, which are extremely popular because they provide users with direct and natural modeling formalism. Indeed, timed-automata can be verified by a sound and complete "region abstraction" to finite graphs [1, 22, 23], but working directly with these graphs is not convenient. A similar abstraction possibly exists for our framework, although it is not immediately clear or obvious how to define an abstraction with multiple agents, especially since they can stop "in the middle" of edges, and one can speak of their relative positions. Many forms of abstraction are not sound/complete. Thus, they do not allow one to map the border between decidability and undecidability as we do. Also, because an abstraction "throws away information", it must be tailor-made depending on the property to be verified.

Future Work. As mentioned above, one possible task is to try and come up with a form of "region abstraction" of our framework that is both sound and complete.

It is interesting to note that one can modify Theorem 1 to the case that the agent has access to absolute positioning with some error (i.e., $\pm\epsilon$, as would be provided by a real-world GPS system). However, it is not clear if it can be modified to the case of an agent that has no global positioning but can measure its step size with some error. This suggests that one may come up with a model for such agents that has a decidable parameterised verification problem (with or without relative positioning).

Two other natural directions of future work are the following. First, one can investigate in practice the actual performance of the presented framework by implementing the proposed algorithms. Although the complexity is PSPACE in general, one can hope — as it is with many algorithms in formal verification — that in many natural cases the algorithm may exhibit acceptable performance. Another possible direction is to try to extend this framework to get decidability results (with reasonable complexity) for multi-robot system scenarios that are rich enough to capture protocols found in the distributed computing literature, e.g., [5, 25, 26, 38].

Acknowledgments

B. Aminof and F. Zuleger are supported by the Austrian National Research Network S11403-N23 (RiSE) of FWF and by WWTF through grant ICT12-059. S. Rubin is a Marie Curie fellow of the Istituto Nazionale di Alta Matematica. A. Murano is partially supported by the GNCS 2016 project: Logica, Automi e Giochi per Sistemi Auto-adattivi.

REFERENCES

- [1] R. Alur, C. Courcoubetis, and D. L. Dill. Model-checking in dense real-time. *Inf. Comput.*, 104(1):2–34, 1993.
- [2] B. Aminof, A. Murano, S. Rubin, and F. Zuleger. Verification of asynchronous mobile-robots in partially-known environments. In *PRIMA*, LNCS 9387, pages 185–200, 2015.
- [3] E. M. Barrameda, S. Das, and N. Santoro. Uniform dispersal of asynchronous finite-state mobile robots in presence of holes. In *ALGOSENSORS*, LNCS 8243, pages 228–243, 2013.
- [4] M. A. Bender, A. Fernandez, D. Ron, A. Sahai, and S. Vadhan. The power of a pebble: Exploring and mapping directed graphs. *I&C*, 176(1):1–21, 2002.
- [5] M. A. Bender and D. K. Slonim. The power of team exploration: Two robots can learn unlabeled directed graphs. Technical report, MIT, 1995.
- [6] R. Bloem, S. Jacobs, A. Khalimov, I. Konnov, S. Rubin, H. Veith, and J. Widder. *Decidability of Parameterized Verification*, volume 6 (1) of *Synthesis Lectures on Distributed Computing Theory*. 2015.
- [7] A. Blum, P. Raghavan, and B. Schieber. Navigating in unfamiliar geometric terrain. *SIAM J. Comput.*, 26(1):110–137, 1997.
- [8] M. Blum and C. Hewitt. Automata on a 2-dimensional tape. *SWAT (FOCS)*, pages 155–160, 1967.
- [9] R. Brafman, J. Latombe, Y. Moses, and Y. Shoham. Applications of a logic of knowledge to motion planning under uncertainty. *J.ACM*, 44(5):633–668, 1997.
- [10] W. Burgard, M. Moors, C. Stachniss, and F. Schneider. Coordinated multi-robot exploration. *IEEE Transactions on Robotics*, 21(3):376–386, 2005.
- [11] A. Canedo-Rodríguez, V. Alvarez-Santos, C. V Regueiro, X. M. Pardo, and R. Iglesias. Multi-agent system for fast deployment of a guide robot in unknown environments. *Journal of Physical Agents*, 6(1):31–41, 2012.
- [12] K. Cheng and P. Dasgupta. Coalition game-based distributed coverage of unknown environments by robot swarms. In *AAMAS*, pages 1191–1194, 2008.
- [13] E. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT, 2002.
- [14] R. Cohen, P. Fraigniaud, D. Ilcinkas, A. Korman, and D. Peleg. Label-guided graph exploration by a finite automaton. In *Automata, Languages and Programming*, LNCS 3580, pages 335–346. 2005.
- [15] S. Das. Mobile agents in distributed computing: Network exploration. *Bull. EATCS*, 109:54–69, 2013.
- [16] G. De Giacomo, P. Felli, F. Patrizi, and S. Sardiña. Two-player game structures for generalized planning and agent composition. In *AAAI*, 2010.
- [17] S. Demri and D. D’Souza. An automata-theoretic approach to constraint LTL. *Inform. and Comp.*, 205(3):380 – 415, 2007.
- [18] S. Demri and R. Gascon. Verification of qualitative Z constraints. *Theor. Comput. Sci.*, 409(1):24–40, 2008.
- [19] A. Dessmark and A. Pelc. Optimal graph exploration without good maps. In *ESA*, LNCS 2461, pages 374–386. Springer, 2002.
- [20] K. Diks, P. Fraigniaud, E. Kranakis, and A. Pelc. Tree exploration with little memory. *Journal of Algorithms*, 51(1):38–63, 2004.
- [21] J. Engelfriet and H. J. Hoogeboom. Nested pebbles and transitive closure. In *STACS*, pages 477–488, 2006.
- [22] M. Faella, S. La Torre, and A. Murano. Dense real-time games. In *LICS*, pages 167–176. IEEE Computer Society, 2002.
- [23] M. Faella, S. La Torre, and A. Murano. Automata-theoretic decision of timed games. *Theor. Comput. Sci.*, 515:46–63, 2014.
- [24] A. Felner, R. Stern, and S. Kraus. PHA*: performing A* in unknown physical environments. In *AAMAS*, pages 240–247. ACM, 2002.
- [25] P. Flocchini, G. Prencipe, and N. Santoro. Computing by mobile robotic sensors. In *Theoretical Aspects of Distributed Computing in Sensor Networks*, EATCS, pages 655–693. 2011.
- [26] P. Flocchini, G. Prencipe, and N. Santoro. *Distributed Computing by Oblivious Mobile Robots*. Synthesis Lectures on Distributed Computing Theory. 2012.
- [27] P. Fraigniaud, D. Ilcinkas, G. Peer, A. Pelc, and D. Peleg. Graph exploration by a finite automaton. *Theoretical Computer Science*, 345:331–344, 2005.
- [28] L. Gasieniec and T. Radzik. Memory efficient anonymous graph exploration. In *Graph-Theoretic Concepts in Computer Science*, LNCS 5344, pages 14–29. Springer, 2008.
- [29] A. Gilboa, A. Meisels, and A. Felner. Distributed navigation in an unknown physical environment. In *AAMAS*, pages 553–560, 2006.
- [30] F. Hoffmann. One pebble does not suffice to search plane labyrinths. In *Fundamentals of Computation Theory*, LNCS 117, pages 433–444. 1981.
- [31] T. Hsiang, E. M. Arkin, M. A. Bender, S. P. Fekete, and J. S. B. Mitchell. Algorithms for rapidly dispersing robot swarms in unknown environments. In *WAFR*, Springer Tracts in Advanced Robotics, vol. 7, pages 77–94. Springer, 2002.
- [32] Y. Hu and G. De Giacomo. Generalized planning: Synthesizing plans that work for multiple environments. In *IJCAI*, pages 918–923, 2011.
- [33] K. Hungerford, P. Dasgupta, and K. R. Guruprasad. Distributed, complete, multi-robot coverage of initially unknown environments using repartitioning. In *AAMAS*, pages 1453–1454. IFAAMAS, 2014.
- [34] A. Khalimov, S. Jacobs, and R. Bloem. PARTY parameterized synthesis of token rings. In *CAV*, LNCS 8044, pages 928–933, 2013.
- [35] A. Khalimov, S. Jacobs, and R. Bloem. Towards efficient parameterized synthesis. In *VMCAI*, LNCS 7737, pages 108–127, 2013.
- [36] P. Kouvaros and A. Lomuscio. Automatic verification of parameterised multi-agent systems. In *AAMAS*, pages 861–868, 2013.
- [37] P. Kouvaros and A. Lomuscio. A counter abstraction technique for the verification of robot swarms. In *AAAI*, pages 2081–2088, 2015.
- [38] E. Kranakis, D. Krizanc, and S. Rajsbaum. Mobile agent rendezvous: A survey. In *SIROCCO*, LNCS 4056, pages 1–9, 2006.
- [39] E. Kranakis, D. Krizanc, and S. Rajsbaum.

- Computing with mobile agents in distributed networks. In *Handbook of Parallel Computing: Models, Algorithms, and Applications. Chapter 8*. 2007.
- [40] N. A. Lynch. *Distributed Algorithms*. Morgan Kaufmann, 1996.
 - [41] D. K. M. Blum. On the power of the compass (or, why mazes are easier to search than graphs). In *FOCS*, pages 132–142, 1978.
 - [42] M. L. Minsky. *Computation: finite and infinite machines*. Prentice-Hall, Inc., 1967.
 - [43] C. H. Papadimitriou and M. Yannakakis. Shortest paths without a map. *Theor. Comput. Sci.*, 84(1):127–150, 1991.
 - [44] N. Rao, S. Karet, W. Shi, and S. Iyengar. *Robot navigation in unknown terrains: Introductory survey of non-heuristic algorithms*. Jul 1993.
 - [45] S. Rubin. Parameterised verification of autonomous mobile-agents in static but unknown environments. In *AAMAS*, pages 199–208, 2015.
 - [46] K. Senthilkumar and K. Bharadwaj. Multi-robot exploration and terrain coverage in an unknown environment. *Robotics and Autonomous Systems*, 60(1):123–132, 2012.

Prompt Alternating-Time Epistemic Logics

Benjamin Aminof

Technische Universität Wien

Aniello Murano and Sasha Rubin

Università di Napoli "Federico II"

Florian Zuleger

Technische Universität Wien

Abstract

In temporal logics, the operator F expresses that at some time in the future something happens, e.g., a request is eventually granted. Unfortunately, there is no bound on the time until the eventuality is satisfied which in many cases does not correspond to the intuitive meaning system designers have, namely, that F abstracts the idea that there is a bound on this time although its magnitude is not known. An elegant way to capture this meaning is through Prompt-LTL, which extends LTL with the operator F_P (“prompt eventually”). We extend this work by studying alternating-time epistemic temporal logics extended with F_P .

We study the model-checking problem of the logic Prompt-KATL*, which is ATL* extended with epistemic operators and prompt eventually. We also obtain results for the model-checking problem of some of its fragments. Namely, of Prompt-KATL (ATL with epistemic operators and prompt eventually), Prompt-KCTL* (CTL* with epistemic operators and prompt eventually), and finally the existential fragments of Prompt-KATL* and Prompt-KATL.

Introduction

Alternating-time temporal logics are expressive tools for reasoning about multi-agent systems (Alur, Henzinger, and Kupferman 2002; van der Hoek and Wooldridge 2002; Chatterjee, Henzinger, and Piterman 2010; Mogavero et al. 2014). These powerful logics allow one to express individual or common goals of the agents throughout time, as well as specify the interactions among the agents (cooperation or adversarial). *Model checking* (Clarke and Emerson 1981; Queille and Sifakis 1981) specifications written in these logics allows one to verify the correct behavior of multi agent systems using recently developed practical automatic tools (Lomuscio, Qu, and Raimondi 2009; Čermák et al. 2014; Čermák, Lomuscio, and Murano 2015).

A pioneering logic in this field is *Alternating-Time Temporal Logic* (ATL*) and its fragments ATL (Alur, Henzinger, and Kupferman 2002), and CTL* (Emerson and Halpern 1986). ATL* formulas are usually interpreted over *concurrent game structures* (CGS), which are labeled-state transition-systems with the ability of modeling the interaction among agents. For example, in a system with multiple

agents and shared resources, the fact that a set of agents A can ensure that, regardless of the actions of the other agents, every request to access a resource is eventually granted, can be expressed by the ATL* formula $\langle\!\langle A \rangle\!\rangle G(req \rightarrow F grant)$.

A crucial shortcoming in real-life temporal-logic verification, deeply ingrained in the definition of linear-temporal logic (LTL), is that the satisfaction of a formula like $F grant$ implies no *a priori* bound on when the *grant* occurs. I.e., the system may admit executions with longer and longer delays before the *grant*, and yet still satisfy the formula $F grant$. Replacing the above formula with a formula specifying that the *grant* should occur within some fixed number of steps (say 3) is usually not an option, since one usually does not know the maximal delay that should be expected. This fact, that the F operator of temporal-logics fails to capture the intuitive meaning of “within some bounded amount of time” has motivated the introduction of an extension of LTL, called “prompt-LTL”, in (Kupferman, Piterman, and Vardi 2009; Alur et al. 2001) that includes a new operator F_P called “prompt eventually”. The semantics of $F_P \phi$ is such that it is satisfied only if there is some bound k , which is shared by all behaviours/computations of the system, such that whenever $F_P \psi$ should hold at some point along a computation then ψ holds within at most k steps. (Kupferman, Piterman, and Vardi 2009) goes on to show that prompt-LTL model checking is not more costly and slightly more complicated than LTL-model checking. It is important to note that prompt-LTL formulas are in positive normal form (i.e., with negations pushed all the way to the atoms), but it does not include the dual operator G_P of the F_P operator (however, the operator G is included). As argued in (Kupferman, Piterman, and Vardi 2009), on the one hand G_P is less useful than F_P (its meaning is that there is some global bound k such that whenever $G_P \psi$ should hold then ψ holds for at least k steps, and we do not care afterwards), and on the other hand adding it to the logic seriously complicates the decision procedures.

Since ATL* inherits its temporal operators from LTL it is natural to consider extending it with the F_P operator, thus allowing one to specify that eventualities should not be delayed for an unbounded number of steps¹. We believe that, in a multi-agent setting, the need for the F_P operator is ar-

¹It is an intriguing open question whether one can write in ATL* (or CTL*) a formula that is equivalent to F_P .

guably even more natural than for closed single-agent systems (for which LTL is suited) as it allows one to specify that certain agents should not have the power to unboundedly delay other agents. Hence, we extend ATL* to include the F_P operator. We combine this with the extension of ATL* with epistemic operators that allow one to express what different agents know in a setting with imperfect information.

Reasoning about agents' strategies in open system verification may require to act under partial information about the states of the system (Aminof, Murano, and Vardi 2007; Jamroga and Ågotnes 2007; Aminof et al. 2013; Bulling and Jamroga 2014; Huang and van der Meyden 2014; Jamroga and Murano 2015). In many practical scenarios such as web-banking attacks, card games, market auctions, etc., agents have indeed a limited observability about the state content. One may think of states containing some private information that are visible only to a (possibly empty) subset of players (Reif 1984; Kupferman and Vardi 1997a). Each agent chooses his strategy based on what he can observe. To work with incomplete information systems, the syntax and the semantics of ATL* has been properly extended with epistemic operators (van der Hoek and Wooldridge 2002). The resulting logic is known as KATL*, sometimes simply called ATL* with *imperfect information*.

Our contribution We address the question of verifying prompt branching-time specifications in multi-agent systems for the first time. We present an extension of KATL* with the prompt eventually temporal operator, called Prompt-KATL*. We study the model-checking problem of this logic and some of its fragments. Namely, of Prompt-KATL (ATL with epistemic operators and prompt eventually), Prompt-KCTL* (CTL* with epistemic operators and prompt eventually), and finally the existential fragments of Prompt-KATL* and Prompt-KATL. We show that, for the case of perfect information, model-checking is decidable and not harder than for these logics without the prompt eventually operator. For the case of imperfect information, note that model checking of ATL*, and even ATL, over concurrent game structures with more than two agents and imperfect information is undecidable ((Pnueli and Rosner 1989; Dima and Tiplea 2011)). Moreover, by (Vester 2013), this is already the case for the existential fragment of ATL. However, we show that the imperfect information case is always decidable for Prompt-KCTL*, and for Prompt-KATL* and Prompt-KATL it is decidable in the following two cases: (i) the players are constrained to memoryless strategies, or (ii) the players use memory-full but cooperative strategies and one restricts to the existential fragments of Prompt-KATL* and Prompt-KATL. Furthermore, in all cases, the complexity of our procedures is as good as for the non-prompt version of these logics.²

Related work In (Alur et al. 2001), the authors introduce a parameterised extension of LTL in which the temporal operators are associated with variables in order to count the

²Except for the case of Prompt-KATL with memoryless strategies for which we only show membership in PSPACE.

steps between successive occurrences of different events. A fragment of this logic is closely studied in (Kupferman, Piterman, and Vardi 2009), i.e., the extension of LTL by the prompt eventuality operator F_P , called *Prompt LTL*. In (Almagor, Hirshfeld, and Kupferman 2010) the automata-theoretic counterpart of the F_P operator has been also introduced and studied.

Only recently has prompt LTL been studied outside the realm of closed systems. (Zimmermann 2013) studies two-player turn-based games of perfect information with respect to prompt LTL. (Chatterjee, Henzinger, and Horn 2009) lift the prompt semantics to ω -regular games, under the parity winning condition, by introducing *finitary parity* games. They make use of the concept of “*distance*” between positions in a play that refers to the number of edges traversed in the game arena. The classical parity winning condition is then reformulated to take into consideration only those states occurring with a bounded distance. This idea has also been generalised to deal with more involved prompt parity conditions (Fijalkow and Zimmermann 2012; Mogavero, Murano, and Sorrentino 2013). Finally, from a practical point of view, one can see connections with bounded model checking of open systems. Indeed, a central question there is whether a model satisfies a given formula by looking at the model up to depth k . We refer to (Huang, Luo, and Van Der Meyden 2011) for an overview.

Due space limitation, most of the proofs are just sketched.

Definitions

Basic Notation

We denote the set of integers by \mathbb{N} , and write $\mathbb{N}_0 := \mathbb{N} \cup \{0\}$. For a set Γ , we write Γ^ω (resp. Γ^*) for the set of infinite (resp. finite) sequences (also called words) of elements in Γ , and Γ^+ for the non-empty finite sequences. We count positions in a sequence starting with 0, and write w_i for the i 'th element (called *letter*) of a word w . The length (in $\mathbb{N}_0 \cup \{\infty\}$) of w is written $|w|$. The suffix $w_i w_{i+1} \dots$ of w is written $w_{\geq i}$ and $w_{\leq i}$ is the prefix $w_0 \dots w_i$ of w . We usually write a instead of the singleton set $\{a\}$. Given $n \in \mathbb{N}$, we consider a function f with domain $\{1, \dots, n\}$ as a vector with n coordinates. Thus we may write $f = (f_1, \dots, f_n)$, and use $f(i)$ and f_i interchangeably.

Game Structures

As for KATL*, models of Prompt-KATL* are *Imperfect Information Concurrent Game Structures (iCGS)*, i.e., structures of the form $S = \langle Ag, AP, Act, S, \lambda, \delta, \{\sim_a : a \in Ag\} \rangle$ where:

- Ag is a finite non-empty set of *agents* (also called *players*);
- AP is a finite non-empty set of atoms; Act is a finite non-empty set of *actions* for the agents;
- S is the set of *states* of the game structure;
- $\lambda : S \rightarrow 2^{AP}$ is a *labeling* function that assigns to a state s the set $\lambda(s)$ of atoms that hold in that state;
- $\delta : S \times Act^{Ag} \rightarrow S$ is a *transition* function that assigns to every state, and every choice of actions — one for each agent — a successor state;

- and $\sim_a \subseteq S \times S$ is an equivalence relation representing the imperfect information of agent a , i.e., $s \sim_a s'$ means that agent a cannot distinguish between s and s' .

The equivalence-classes of \sim_a are called the *observation sets* of agent a . The set Act^{Ag} is called the set of *decisions*. An iCGS is called *finite* if the set S is finite. An iCGS has *perfect information* if for every $a \in Ag$ we have that \sim_a is the equality relation, i.e., if $s \sim_a s'$ implies that $s = s'$. In this case it may be written CGS.

Observe that we assume that all agents use the same set of actions Act . However, it is sometimes convenient to assume that each agent a uses only some non-empty subset $Act_a \subseteq Act$ of actions (one can simply define the transition relation to have all actions in $Act \setminus Act_a$ duplicate the effect of some action in Act_a)³.

Computations and Strategies. A path in S is a finite or infinite sequence $\pi_0\pi_1\cdots \in S^\omega \cup S^+$ such that for all i there exists a decision $d \in Act^{Ag}$ such that $\pi_{i+1} = \delta(\pi_i, d)$. We call finite paths *histories*, and infinite ones *computations* or *plays*. The set of computations in S is written $cmp(S)$, and the set of computations in S that start with s is written $cmp(S, s)$. We define $hist(S)$ and $hist(S, s)$ similarly.

A *strategy* (for a single agent) is a function $\sigma : hist(S) \rightarrow Act$. For a non-empty set $A \subseteq Ag$ of agents, and a strategy σ_a for each $a \in A$, write $\Sigma_A := \{\sigma_a : a \in A\}$ for the set of strategies. A path π is *consistent with* Σ_A if it can be obtained by having the agents in A follow their strategies in Σ_A , i.e., if for every position π_i of π there exists $d \in Act^{Ag}$ such that: i) $\pi_{i+1} \in \delta(\pi_i, d)$ and; ii) for every $a \in A$, $d(a) = \sigma_a(\pi_{\leq i})$. The set of computations starting with s that are consistent with Σ_A , written $out(s, \Sigma_A)$, is called the set of *outcomes* of Σ_A from s .

For $A \in Ag$, we derive the following equivalence relations: $\sim_A := \cap_{a \in A} \sim_a$, and $\sim_A^C := (\cup_{a \in A} \sim_a)^*$, where $*$ denotes the transitive closure (with respect to composition). Note that $\sim_{\{a\}} = \sim_a$, and we use these interchangeably.

Extend \sim_A to histories point-wise: if hs and $h's'$ are histories and $h \sim_A h'$ and $s \sim_A s'$ then $hs \sim_A h's'$. A strategy σ is *observational for agent a* if for all $h \sim_a h'$, we have $\sigma(h) = \sigma(h')$. A set Σ_A of strategies for the agents in A is *cooperatively observational* if for all $h \sim_A h'$ and $a \in A$, we have $\sigma_a(h) = \sigma_a(h')$.⁴

2-Player Games. The proofs of Propositions 3 and 4 use the following notions. A *2-player concurrent game* is a pair $\langle S, \Upsilon \rangle$ where S is an iCGS (called an *arena*) with two players (usually $Ag = \{0, 1\}$), and $\Upsilon \subseteq cmp(S)$ is a *goal*. A play $\pi \in cmp(S)$ is *won* by Player 0 if $\pi \in \Upsilon$, and is won by Player 1 otherwise. Given $s \in S$, an observational strategy σ_i for Player $i \in \{0, 1\}$ is called *winning from s* (and we say that Player i *wins from s*) if all plays starting in s that are consistent with σ_i are won by Player i (i.e., if $out(s, \sigma_i) \subseteq \Upsilon$). An LTL formula ψ induces a goal $\Upsilon := \{\pi \in cmp(S) \mid$

³More formally, require that for every agent a , there is some action $\alpha \in Act_a$, such that for every $d \in Act^{Ag}$ and $s \in S$, we have: if $d(a) \in Act \setminus Act_a$ then $\delta(s, d) = \delta(s, d')$, where d' is obtained from d by letting $d'(a) = \alpha$.

⁴Agents using cooperative strategies are sometimes referred to as having *distributive knowledge*.

$\pi \models \psi\}$, and we usually just say that the game has goal ψ . If Player 0 wins from s in the game with goal ψ we say that he *can enforce* ψ from s .

Syntax of Prompt-KATL*.

Fix a finite set of *atomic propositions (atoms)* AP, and a finite set of *agents* Ag. The *Prompt-KATL** state (φ) and path (ψ) formulas over AP and Ag are built using the following context-free grammar:

$$\begin{aligned}\varphi ::= & p \mid \neg p \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \langle\!\langle A \rangle\!\rangle \psi \mid [[A]] \psi \mid \\ & \mathbb{K}_a \varphi \mid \mathbb{D}_A \varphi \mid \mathbb{C}_A \varphi \mid \widetilde{\mathbb{K}}_a \varphi \mid \widetilde{\mathbb{D}}_A \varphi \mid \widetilde{\mathbb{C}}_A \varphi\end{aligned}$$

and

$$\psi ::= \varphi \mid \psi \vee \psi \mid \psi \wedge \psi \mid X \psi \mid \psi U \psi \mid \psi R \psi \mid F_P \psi$$

where p varies over AP, A varies over subsets of Ag, and a over elements of Ag. The class of *Prompt-KATL** formulas is the set of state formulas generated by the grammar.

The *temporal operators* are X (next), U (until), R (releases, the dual of until) and F_P (prompt eventually); the *strategy quantifiers* are $\langle\!\langle A \rangle\!\rangle$, $[[A]]$, where $\langle\!\langle A \rangle\!\rangle \psi$ is read “the agents in A can enforce ψ ”, and its dual $[[A]] \psi$ is read “the agents in A cannot avoid ψ ”; and the *epistemic operators* are \mathbb{K}_a (agent a knows that), \mathbb{D}_A (the agents in A distributively know that), and \mathbb{C}_A (amongst the agents in A it is common knowledge that), as well as the dual operators $\widetilde{\mathbb{K}}_a$, $\widetilde{\mathbb{D}}_A$, $\widetilde{\mathbb{C}}_A$. Observe that, as discussed in the introduction, we do not include a dual operator to F_P . Also note that if one removes the prompt eventually operator then the grammar above generates exactly all the formulas of KATL* in positive normal form (i.e., with negations pushed all the way to the atoms).

We have the usual syntactic sugar: we write $F \varphi$ (eventually) instead of true $U \varphi$; (globally) $G \varphi$ instead of false $R \varphi$; E (exists) instead of $\langle\!\langle Ag \rangle\!\rangle$, and A (for all) instead of $[[Ag]]$; \mathbb{E} (everybody knows) instead of $\wedge_{a \in Ag} \mathbb{K}_a$ (and its dual $\widetilde{\mathbb{E}}$ for $\vee_{a \in Ag} \widetilde{\mathbb{K}}_a$). Finally, we use a shorthand for repeated next: X^k (for $k \in \mathbb{N}$) is defined as: $X^1 := X$, and $X^{k+1} := XX^k$.

We now define some important syntactic fragments.

1. Prompt-ATL* formulas consists of the formulas of Prompt-KATL* in which no epistemic operator occurs.
2. Prompt-ATL formulas consists of the formulas of Prompt-ATL* in which every temporal operator is immediately preceded by a strategy quantifier.
3. Prompt-KCTL* is obtained from Prompt-KATL* by only allowing strategy quantifiers of the form E and A.
4. Prompt-LTL is the class of path formulas generated by the grammar above in which no strategy quantifier or epistemic operator appears.
5. The *existential fragments* of Prompt-KATL* and Prompt-KATL consist of those formulas in which the $[[A]]$ quantifier does not occur.

Semantics of Prompt-KATL*.

We first define the semantics of KATL* (i.e., formulas that don't mention the prompt operator F_P), and then define the semantics of Prompt-KATL*.

Semantics of KATL^{*}. The satisfaction relation \models is defined inductively, as usual. Formally, for all $s \in S, p \in AP$:

- $(S, s) \models p$ iff $p \in \lambda(s)$, and $(S, s) \models \neg p$ iff $p \notin \lambda(s)$.
- $(S, s) \models \varphi_1 \wedge \varphi_2$ iff $(S, s) \models \varphi_1$ and $(S, s) \models \varphi_2$.
- $(S, s) \models \varphi_1 \vee \varphi_2$ iff $(S, s) \models \varphi_1$ or $(S, s) \models \varphi_2$.
- $(S, s) \models \langle\langle A \rangle\rangle \psi$ iff there exists a set of observational strategies Σ_A , one for each agent in A , s.t. $(S, \pi) \models \psi$ for all $\pi \in out(s, \Sigma_A)$.
- $(S, s) \models [[A]]\psi$ iff for every set of observational strategies Σ_A , one for each agent in A , there is a computation $\pi \in out(s, \Sigma_A)$ s.t. $(S, \pi) \models \psi$.
- $(S, s) \models \mathbb{K}_a \varphi$ (resp. $\widetilde{\mathbb{K}}_a \varphi$) iff $(S, s') \models \varphi$ for every s' (resp. some s') s.t. $s' \sim_a s$.
- $(S, s) \models \mathbb{D}_A \varphi$ (resp. $\widetilde{\mathbb{D}}_A \varphi$) iff $(S, s') \models \varphi$ for every s' (resp. some s') s.t. $s' \sim_A s$.
- $(S, s) \models \mathbb{C}_A \varphi$ (resp. $\widetilde{\mathbb{C}}_A \varphi$) iff $(S, s') \models \varphi$ for every s' (resp. some s') s.t. $s' \sim_A^C s$.

and for all $\pi \in cmp(S)$:

- $(S, \pi) \models \varphi$, for φ a state formula, iff $(S, \pi_0) \models \varphi$.
- $(S, \pi) \models \psi_1 \wedge \psi_2$ iff $(S, \pi) \models \psi_1$ and $(S, \pi) \models \psi_2$.
- $(S, \pi) \models \psi_1 \vee \psi_2$ iff $(S, \pi) \models \psi_1$ or $(S, \pi) \models \psi_2$.
- $(S, \pi) \models X \psi$ iff $(S, \pi_{\geq 1}) \models \psi$.
- $(S, \pi) \models \psi_1 U \psi_2$ iff there $i \in \mathbb{N}$ such that $(S, \pi_{\geq i}) \models \psi_2$ and for all $j < i$, $(S, \pi_{\geq j}) \models \psi_1$.
- $(S, \pi) \models \psi_1 R \psi_2$ iff for all i , either $(S, \pi_{\geq i}) \models \psi_2$ or there exists $j < i$ such that $(S, \pi_{\geq j}) \models \psi_1$.

We emphasise two points: strategies have perfect recall, and epistemic operators depend only on the current state and not on the history (the latter appears, e.g., in (Čermák et al. 2014)).

Semantics of Prompt-KATL^{*}. For $k \in \mathbb{N}$, we use the notation $(S, s) \models^k \varphi$ and $(S, \pi) \models^k \psi$ to denote that the formula is satisfied in which every prompt eventuality is fulfilled within at most k steps. Formally: define $(S, s) \models^k \varphi$ and $(S, \pi) \models^k \psi$ inductively, as above,⁵ with the following additional rule:

- $(S, \pi) \models^k F_P \psi$ iff there exists $j \leq k$ such that $(S, \pi_{\geq j}) \models^k \psi$. Say that π models ψ with bound k .

Definition 1. For a Prompt-KATL^{*} state-formula φ and $s \in S$, define $(S, s) \models \varphi$ iff there exists $k \in \mathbb{N}$ such that $(S, s) \models^k \varphi$. Say that φ is satisfied at s .

Linearising branching-formulas and Prompt-LTL

Like CTL^{*}, and ATL^{*} after it, one can think of a Prompt-KATL^{*} path formula ψ over atoms AP as a Prompt-LTL formula $lin(\psi)$ over atoms which are the maximal state subformulas of ψ , as follows (Kupferman, Vardi, and Wolper 2000).

⁵Thus, e.g., $(S, \pi) \models^k \psi_1 U \psi_2$ iff there $i \in \mathbb{N}$ such that $(S, \pi_{\geq i}) \models^k \psi_2$ and for all $j < i$, $(S, \pi_{\geq j}) \models^k \psi_1$.

A formula φ is a *state subformula* of ψ if φ is a state formula as well as a subformula of ψ . A formula φ is a *maximal state subformula* of ψ if $\varphi \neq \psi$, it is a state subformula of ψ , and it is not a proper subformula of any other state subformula of ψ . Let $max(\psi)$ be the set of maximal state subformulas of ψ .

Every Prompt-KATL^{*} path formula ψ can be viewed as a Prompt-LTL formula, call it $lin(\psi)$, whose atoms are elements of $max(\psi)$. Formally:

Definition 2. For a path formula ψ , define $lin(\psi)$ as follows — in each case note that $lin(\psi)$ is a formula over atoms $max(\psi)$:

- $lin(p) := p$ and $lin(\neg p) := \neg p$;
- For $\circ \in \{\langle\langle A \rangle\rangle, [[A]], \mathbb{K}_a, \mathbb{D}_A, \mathbb{C}_A\}$, $lin(\circ \psi) := \circ lin(\psi)$;
- If $\psi = \phi_1 \circ \phi_2$ for $\circ \in \{\vee, \wedge\}$, then $lin(\phi_1 \circ \phi_2)$ is defined to be $lin(\phi_1) \circ lin(\phi_2)$ (note this is well defined since at least one ϕ_i is not a state formula);
- For $\circ \in \{X, F_P\}$, $lin(\circ \phi) := \circ lin(\phi)$;
- For $\circ \in \{U, R\}$, $lin(\psi_1 \circ \psi_2) := lin(\psi_1) \circ lin(\psi_2)$.

For example, if $\psi = (p U \langle\langle A \rangle\rangle F_P q) \vee X \neg p$, then its state subformulas are $\{p, \langle\langle A \rangle\rangle F_P q, q, \neg p\}$, and $max(\psi) = \{p, \langle\langle A \rangle\rangle F_P q, \neg p\}$, and thus $lin(\psi)$ is the Prompt-LTL formula $(\boxed{p} U \langle\langle A \rangle\rangle F_P q) \vee X \boxed{\neg p}$ over the atoms $max(\psi)$ (for illustration we box the subformulas that are treated as atoms).

For an iCGS $S := \langle Ag, AP, Act, S, \lambda, \delta, \{\sim_a : a \in Ag\} \rangle$, and a Prompt-KATL^{*} path formula ψ over AP, define the iCGS $S_\psi := \langle Ag, max(\psi), Act, S, \lambda, \delta_\psi, \{\sim_a : a \in Ag\} \rangle$ with atoms $max(\psi)$ and a labeling $\lambda_\psi : S \rightarrow 2^{max(\psi)}$ defined by letting $\varphi \in \lambda_\psi(s)$ iff $(S, s) \models \varphi$. The next lemma says that a computation π of S satisfies ψ iff, when viewed as a computation of S_ψ , it satisfies $lin(\psi)$:

Lemma 1. For every iCGS S , Prompt-KATL^{*} path formula ψ over atoms AP, and computation π of S , we have that $(S, \pi) \models \psi$ if and only if $(S_\psi, \pi) \models lin(\psi)$.

Proof. The proof is by induction on the structure of ψ , using the following inductive hypothesis on the subformulas ϕ of ψ (that are not proper subformulas of any formula in $max(\psi)$): for every position $i \in \mathbb{N}$ of π , we have that $(S, \pi_{\geq i}) \models \phi$ iff $(S_\psi, \pi_{\geq i}) \models lin(\phi)$. \square

Prompt linear-temporal logic.

We now consider Prompt-LTL in more detail.

Notation. For a path $\pi \in cmp(S)$ and a Prompt-LTL formula ψ , write $\pi \models \psi$ instead of $(S, \pi) \models \psi$. Also, if $v \in (2^{AP})^\omega$ then we abuse notation and write $v \models \psi$.

Recall from the definitions of syntax that we define Prompt-LTL as the syntactic fragment of Prompt-KATL^{*} in which $\langle\langle A \rangle\rangle$ and $[[A]]$ do not occur. Thus, if ψ is a Prompt-LTL formula and S is a CGS, then $(S, s) \models A \psi$ expresses that there is a bound $k \in \mathbb{N}$ such that for every $\pi \in cmp(S)$ starting in s we have that $(S, \pi) \models^k \psi$.⁶

⁶Remark: the syntax in (Kupferman, Piterman, and Vardi 2009) is different. There they write, e.g., $S \models \psi$ instead of $S \models A \psi$.

Proposition 1. (Kupferman, Piterman, and Vardi 2009) The following problem is decidable: given **Prompt-LTL** formula ψ , a finite CGS S , and a state $s \in S$, decide whether $(S, s) \models \mathbf{A} \psi$. Moreover, the complexity is PSPACE in the size of ψ and NLOGSPACE in the size of S .

The previous proposition allows us to deal with universal quantifiers. We now deal with the existential quantifier.

Definition 3. If ψ is a **Prompt-LTL** formula, define $\text{live}(\psi)$ as the LTL formula that results from ψ by replacing every F_P by F .

The following lemma says that $(S, s) \models E\psi$ if and only if $(S, s) \models E\text{live}(\psi)$. The reason, roughly, is as follows. For the forward direction, note that if ψ holds promptly on a computation, then in particular, it holds eventually (this is because our formulas are in positive-normal form and so the prompt eventualities can not be negated). For the reverse direction, use the fact that if S has a computation satisfying $\text{live}(\psi)$ then it has such a computation that is a lasso, i.e., of the form uv^ω (this holds for all LTL formulas, and thus $\text{live}(\psi)$ in particular, (Vardi and Wolper 1994)). In this case, every subformula of ψ that holds in a suffix $\pi_{\geq j}$ of π (with $j \geq |u| + |v|$) also holds in the suffix $\pi_{\geq j-|v|}$. Thus all eventualities hold promptly (i.e., within $|u| + |v|$ steps). The formal proof is in the full version of the paper.

Lemma 2. If $\pi = uv^\omega$ is a lasso, and ψ is a **Prompt-LTL** formula, then the following are equivalent:

- a) $\pi \models \psi$,
- b) $\pi \models \text{live}(\psi)$,
- c) $\pi \models^b \psi$ where $b = |u| + |v|$.

Proof. Clearly c) \rightarrow a). For a) \rightarrow b) an easy induction on ψ shows that for every computation π , and every $k \in \mathbb{N}$, if $\pi \models^k \psi$ then $\pi \models \text{live}(\psi)$. For b) \rightarrow c) we prove, by induction on ψ , that for all $i \in \mathbb{N}$, and all subformulas ψ' of ψ : if $\pi_{\geq i} \models \text{live}(\psi')$ then $\pi_{\geq i} \models^b \psi'$ (to complete the proof take $i = 0$ and $\psi' = \psi$). The only non-trivial case is $\psi' = F_P \psi_1$. Thus, suppose $\pi_{\geq i} \models \text{live}(F_P \psi_1)$. Then $\pi_{\geq i} \models F \text{live}(\psi_1)$, and so there exists $j \geq i$ such that $\pi_{\geq j} \models \text{live}(\psi_1)$. There are two cases.

Suppose $j < |u|$. By induction, $\pi_{\geq j} \models^b \psi_1$, and so $\pi_{\geq i} \models^{\max\{j-i, b\}} F_P \psi_1$. Since $j - i < |u| - i < b$, we have that $\pi_{\geq i} \models^b F_P \psi_1$, as required.

Suppose $j \geq |u|$. Pick n_0 so that $\max\{i, |u|\} \leq j - |v|n_0 \leq \max\{i, |u|\} + |v|$. Since $\pi = uv^\omega$, we have that $\pi_{\geq j-|v|n_0}$ is equal to π_j . Thus $\pi_{\geq j-|v|n_0} \models \text{live}(\psi_1)$, and so by induction $\pi_{\geq j-|v|n_0} \models^b \psi_1$, and so $\pi_{\geq i} \models^{\max\{j-|v|n_0-i, b\}} F_P \psi_1$. Since $j - |v|n_0 - i \leq \max\{i, |u|\} + |v| - i \leq |u| + |v| = b$, we have $\pi_{\geq i} \models^b F_P \psi_1$, as required. \square

We now get:

Proposition 2. For every **Prompt-LTL** formula ψ , CGS S , and state $s \in S$, we have that $(S, s) \models E\psi$ if and only if $(S, s) \models E\text{live}(\psi)$. Moreover, the cost of checking this fact is PSPACE in the size of ψ and NLOGSPACE in the size of S .

Proof. Since $\text{live}(\psi)$ is an LTL formula, we have that $(S, s) \models E\text{live}(\psi)$ if and only if there is a lasso $\pi = uv^\omega$ starting in s such that $(S, \pi) \models \text{live}(\psi)$ (Vardi and Wolper 1994). By Lemma 2 this is equivalent to $(S, \pi) \models^b \psi$ where $b = |u| + |v|$. By definition of \models^b , this is equivalent to $(S, s) \models E\psi$. The complexity follows from the model-checking complexity of LTL (Sistla and Clarke 1985; Kupferman, Piterman, and Vardi 2009). \square

The final lemma of this section will be used in the proof of Proposition 4 (used as part of Theorem 4 on co-operative strategies). The lemma relies on the alternating-color technique from (Kupferman, Piterman, and Vardi 2009), that we now describe. Given a computation π of an iCGS S , add a new atomic proposition red , and imagine that states in which red holds are colored red, and otherwise white. Given a **Prompt-LTL** formula ψ , derive from it an LTL formula $\text{col}(\psi)$ by replacing every subformula of the form $F_P \phi$ with a subformula that says that ϕ holds before the color changes twice. Now, if we can come up with a coloring of π in which the colors alternate fast enough (say every k steps or less), such that the colored version of π models $\text{col}(\psi)$, then we can deduce that π models ψ with bound $2k$; conversely, if π models ψ with bound k , then by changing colors every k steps we can ensure that $\text{col}(\psi)$ is satisfied. This allows us to replace reasoning about **Prompt-LTL** formulas with reasoning about colorings and LTL formulas. We now formally describe the required elements for applying this reasoning.

For $w \in (2^{\text{AP} \cup \{\text{red}\}})^\omega$, a *block* is a maximal subword $w_i \dots w_j$ of w such that $\text{red} \in w_l$ for all $l \in [i, j]$ or $\text{red} \notin w_l$ for all $l \in [i, j]$. For $k \in \mathbb{N}$, say that w is *k-coloured* if the size of every block of w has length at most k . Say that $w \in (2^{\text{AP} \cup \{\text{red}\}})^\omega$ is a *colouring* of $v \in (2^{\text{AP}})^\omega$ if and only if for every i , $w_i \cap \text{AP} = v_i$. Say that w is a *k-colouring* of v if w is a colouring of v and is *k-coloured*.

For a **Prompt-LTL** formula ψ , let ψ' be the LTL formula obtained by replacing every subformula of ψ , of the form $F_P \phi$, by $\text{within}(\phi) := (\text{red} \cup (\neg \text{red} \cup \phi)) \vee (\neg \text{red} \cup (\text{red} \cup \phi))$, which states that ϕ should hold at some point within this color block or the next. Let $\text{col}(\psi) := \psi' \wedge \rho$, where $\rho := G F(\text{red} \wedge X \neg \text{red})$ states that the colors change infinitely often. It is important to note that while $\text{col}(\psi)$ is exponentially larger than ψ , the number of subformulas it has is linear in the number of subformulas of ψ .

Lemma 3. For every **Prompt-LTL** formula ψ , word $v \in (2^{\text{AP}})^\omega$ and $k \in \mathbb{N}$: (i) if there is a *k-colouring* w of v such that $w \models \text{col}(\psi)$ then $v \models^{2k} \psi$; (ii) if $v \models^k \psi$ then there is a *k-colouring* w of v such that $w \models \text{col}(\psi)$, moreover, w can be chosen with all blocks of size exactly k .

Proof. Although this lemma is easily extractable from (Kupferman, Piterman, and Vardi 2009), for completeness we provide the proof. For (i), given a *k-colouring* w of v , if $w \models \text{col}(\psi)$ then the result (that $w \models^{2k} \psi$) follows by induction on ψ and the following observation: for every $i \in \mathbb{N}$ and every subformula $F_P \phi$ of ψ , we have $w_{\geq i} \models \text{within}(\phi)$ implies that $v_{\geq i} \models^{2k} F_P \phi$.

For (ii), if $v \models^k \psi$ then colour v by blocks of size exactly k to get w . The result (that $w \models \text{col}(\psi)$) follows by induc-

tion on ψ and the following observation: for every $i \in \mathbb{N}$ and every subformula $F_P \phi$ of ψ , we have $v_{\geq i} \models^k F_P \phi$ implies that $w_{\geq i} \models \text{within}(\phi)$. \square

Deciding the model-checking problems

The model-checking problem for a logic \mathcal{L} is the following: given a formula φ from \mathcal{L} and a finite iCGS S , decide whether $S \models \varphi$.

Fact 1. *Model checking ATL* is undecidable over concurrent game structures with $|Ag| \geq 3$, and imperfect information, already for the existential fragment (Pnueli and Rosner 1989; Dima and Tiplea 2011).*

Thus, also model checking Prompt-KATL* is undecidable for three or more agents with imperfect information. We show that one can regain decidability (and we provide the optimal complexity) in four ways: (i) restricting to iCGS with perfect information, or (ii) restricting to path quantifiers (instead of strategy quantifiers), or (iii) restricting to memoryless strategies, or (iv) restricting to cooperative strategies and the existential fragment.

Prompt-ATL* with perfect information

Proposition 3. *The following problems are decidable: given a Prompt-LTL formula ψ , a finite (perfect-information) CGS S , a set of agents $A \subseteq Ag$, and state $s \in S$, decide whether $(S, s) \models \langle\langle A \rangle\rangle \psi$; and, similarly, decide whether $(S, s) \models [[A]]\psi$. Moreover, the complexity of these problems is 2EXPTIME in the number of subformulas of ψ , and polynomial in the size of S .*

Proof. We will reduce each question to the problem of deciding if a given player has a winning strategy in Prompt-LTL turn-based games. The latter are solvable in 2EXPTIME (Zimmermann 2013).

We first consider the case of $\langle\langle A \rangle\rangle \psi$. In the first step, build a two-player arena G such that (\star) : $(S, s) \models \langle\langle A \rangle\rangle \psi$ if and only if there exists $k \in \mathbb{N}$ such that Player 0 can enforce (in G) the LTL goal ψ_k from s , where ψ_k is formed from ψ by replacing every subformula of the form $F_P \phi$ by $\bigvee_{i \leq k} X^i \phi$. The idea is that Player 0 corresponds to the coalition A and Player 1 to the coalition $Ag \setminus A$. In more detail: $S = \langle Ag, AP, Act, S, \lambda, \delta \rangle$, define \tilde{G} to be the (perfect information) CGS with two agents, Player 0 and Player 1, $\langle \{0, 1\}, AP, Act_0 \cup Act_1, S, \lambda, \delta_G \rangle$ as follows:

- action sets $Act_0 := Act^A$ and $Act_1 := Act^{Ag \setminus A}$,
- state set S , labeling λ ,
- transition function $\delta_G : S \times Act_0 \times Act_1 \rightarrow S$ that maps $(s, (d_1, d_2)) \mapsto \delta(s, d_1 \otimes d_2)$ where $d_1 \otimes d_2 \in Act_0^{Ag}$ maps a to $d_1(a)$ for $a \in A$ and otherwise (for $a \in Ag \setminus A$) to $d_2(a)$.

It is immediate from the definitions (of G and \models) that (\star) holds. For the next step, recall the following folk fact (\dagger) : Player 0 has a winning strategy in a concurrent game G with goal Υ if and only if Player 0 has a winning strategy in the turn-based game G^{tb} , which simulates G as follows: first, Player 0 moves (thus revealing his chosen action) and then

Player 1 chooses his action and the simulated move is completed. To “skip” the intermediate nodes that G^{tb} introduces, we use the goal Υ^{tb} which is defined to be the set of all computations such that the subsequence consisting of only the even positioned nodes is in Υ . The intuitive reason that this lemma is true is that in the game G^{tb} Player 0 has no new information available to it, and thus it is exactly as hard (or easy) for him to win as in G .⁷

Formally, we build the turn-based game G^{tb} of perfect information with goal Υ_k^{tb} as follows. Let G^{tb} be the 2-player game, over atoms AP , and such that:

- the state set S^{tb} is $S \cup (S \times Act_0)$,
- the labeling function λ^{tb} maps $s \mapsto \lambda(s)$ and $(s, d) \mapsto \emptyset$,
- the transition function $\delta^{tb} : S^{tb} \times Act_0 \times Act_1 \rightarrow S^{tb}$ maps state $s \in S$ and actions (d_0, d_1) to (s, d_0) , and maps a state $(s, d_0) \in S \times Act_0$ and actions (d'_0, d_1) to $\delta(s, d_0 \otimes d_1)$ (for all $s \in S$, $d_0, d'_0 \in Act_0$ and $d_1 \in Act_1$).

Moreover, Υ_k^{tb} consists of those plays whose subsequence consisting of every other node satisfies ψ_k .

Thus, we have reduced our problem to deciding if there exists $k \in \mathbb{N}$ such that Player 0 has a winning strategy in the game G^{tb} with goal Υ_k^{tb} . The latter is the problem of solving a two-player turn-based Prompt-LTL game, which, by (Zimmermann 2013), can be done in 2-EXPTIME. Moreover, that procedure is already able to deal with goals which only look at every second node of the play (so called “blinking semantics”). Thus, the procedure can be easily adapted to decide if there exists $k \in \mathbb{N}$ such that Player 0 has a winning strategy in the game G^{tb} with goal Υ_k^{tb} , which completes the $\langle\langle A \rangle\rangle \psi$ case.

We turn to the $[[A]]\psi$ case. Dually to the case above, build a two-player game H by associating the coalition A with Player 1 and associating the coalition $Ag \setminus A$ with Player 0. Thus, $(S, s) \models [[A]]\psi$ if and only if there exists $k \in \mathbb{N}$ such that Player 1 does not have a winning strategy in the game H with goal ψ_k . Dually again, build H^{tb} in which Player 1 moves first and use the fact (\dagger) to deduce that, for every $k \in \mathbb{N}$, Player 1 does not have a winning strategy in H with goal ψ_k if and only if Player 1 does not have a winning strategy in the game H^{tb} with goal Υ_k^{tb} . Since turn-based games of perfect information with LTL goals (the “blinking semantics” is easily accommodated) are determined (i.e., one of the players has a winning strategy), then this is equivalent to Player 0 having a winning strategy in the game G^{tb} with goal Υ_k^{tb} . But this is the same type of game we had already solved in the case of $\langle\langle A \rangle\rangle \psi$. This completes the $[[A]]\psi$ case. \square

Theorem 1. *The model checking problem for Prompt-ATL* (resp. Prompt-ATL) is 2EXPTIME-complete (resp. PTIME-complete).*

Proof. By (Alur, Henzinger, and Kupferman 2002), the lower bound already holds for ATL* (resp. ATL). For the upper bound, we adapt the marking algorithm for model checking ATL* (Alur, Henzinger, and Kupferman 2002).

⁷However, Player 1 can win in G^{tb} from states in which he can only prevent Player 0 from winning in G , but not ensure he himself wins.

Let φ be a state **Prompt-ATL**^{*} formula, and mark every state $s \in S$ by the state subformulas φ' of φ that are satisfied at s . This is done inductively. The case that φ' is an atom, a negation of an atom, a disjunction or a conjunction is immediate (e.g., if $\varphi' = \varphi_1 \vee \varphi_2$ and s is marked by φ_1 then also mark s by $\varphi_1 \vee \varphi_2$).

The case that $\varphi' = \langle\!\langle A \rangle\!\rangle \psi$ is dealt with as follows. Recall that $lin(\psi)$ is a **Prompt-LTL** formula over atoms $max(\psi)$. By Lemma 1, deciding if $(S, s) \models \langle\!\langle A \rangle\!\rangle \psi$ is equivalent to deciding if $(S_\psi, s) \models \langle\!\langle A \rangle\!\rangle lin(\psi)$. By induction, the satisfaction of all state subformulas of ψ have already been determined. In particular, we have enough information to form S_ψ . By Proposition 3, deciding whether $(S_\psi, s) \models \langle\!\langle A \rangle\!\rangle lin(\psi)$ can be done in 2EXPTIME. The case that $\varphi' = [[A]]\psi$ is similar.

The case of **Prompt-ATL** follows by noting that the formula $lin(\psi)$ is always of constant size since **Prompt-ATL** does not allow temporal operators to be directly nested. \square

Prompt-KCTL^{*}

In the case of **Prompt-KCTL**^{*}, the strategy quantifiers are restricted to E, A. This inherent restriction on the strategy quantifiers results in a decidable model checking problem, also in the presence of imperfect information.

Theorem 2. *Model checking **Prompt-KCTL**^{*} is PSPACE-complete, and the structure complexity is PTIME-complete.*

Proof. The lower-bounds already hold for CTL^{*} (Kupferman, Vardi, and Wolper 2000). For the upper-bounds, let φ be a state **Prompt-KCTL**^{*} formula, and mark every state $s \in S$ by the state subformulas φ' of φ that are satisfied at s . This is done inductively as in Theorem 1. The cases we have to consider are the epistemic operators and the complexity of the path quantifiers. The case that φ' is of the form $\mathbb{K}_a \phi$ is dealt with as follows: mark s by $\mathbb{K}_a \phi$ iff every $s' \sim_a s$ is already marked by ϕ ; similarly, mark s by $\mathbb{D}_A \phi$ iff every $s' \sim_A s$ is already marked by ϕ (the remaining epistemic operators, including the duals, are similar). Each of these steps can be performed in time polynomial in S .

We now discuss the case that φ' is of the form E ψ or A ψ . Recall that $lin(\psi)$ is a **Prompt-LTL** formula over atoms $max(\psi)$, and that $live(\cdot)$ replaces every F_P by F in a **Prompt-LTL** formula. Thus, $live(lin(\psi))$ is an LTL formula over atoms $max(\psi)$. By induction, the satisfaction of all state subformulas of ψ has already been determined. In particular, we have enough information to form the CGS S_ψ (note that since S is a CGS, so is S_ψ). By Lemma 1 note that (†): for $Q \in \{E, A\}$, $(S, s) \models Q\psi$ if and only if $(S_\psi, s) \models Qlin(\psi)$.

For $\varphi' = A\psi$ mark s by A ψ if and only if $(S_\psi, s) \models A lin(\psi)$. To see this is correct use (†).

For $\varphi' = E\psi$, mark s by E ψ if and only if $(S_\psi, s) \models E live(lin(\psi))$. To see that this is correct use (†), Proposition 2, and the fact that $lin(\psi)$ is a **Prompt-LTL** formula over atoms $max(\psi)$.

For the complexity, note that i) there are only a linear number of subformulas φ' of φ , and ii) each case φ' of the algorithm can be done in PSPACE in the size of φ' and NLOGSPACE in the size of S (for the $\varphi' = A\psi$ case

use Proposition 1, and for the $\varphi' = E\psi$ case use the fact that model checking LTL is in PSPACE (Sistla and Clarke 1985)). \square

Prompt-KATL^{*} with memoryless strategies

In this section we prove that model checking **Prompt-KATL**^{*} with memoryless strategies is PSPACE-complete. We begin with the relevant definitions.

A strategy σ is called *memoryless* if for all histories h, h' , and every state s we have $\sigma(hs) = \sigma(h's)$. It is thus common to consider memoryless strategies as functions from states (not histories) to actions.

Define \models_{mem}^k like \models^k except replace the definition of the semantics of the strategy quantifiers to limit the agents to memoryless (observational) strategies as follows:⁸

- $(S, s) \models_{mem}^k \langle\!\langle A \rangle\!\rangle \psi$ (resp. $[[A]]\psi$) iff there exists a set (resp. for all sets) Σ_A of memoryless observational strategies, one strategy for each agent in A , such that for all (resp. for at least one) computation $\pi \in out(s, \Sigma_A)$, we have $(S, \pi) \models^k \psi$.

Define $(S, s) \models_{mem} \varphi$ iff there exists $k \in \mathbb{N}$ such that $(S, s) \models_{mem}^k \varphi$,

Theorem 3. *The following problem is PSPACE-complete: given a **Prompt-KATL**^{*} formula φ and a finite iCGS S , decide whether $S \models_{mem} \varphi$.*

Proof. The lower-bound already holds for CTL^{*} (Kupferman, Vardi, and Wolper 2000). For the upper bound, we use the marking algorithm as we did in Theorem 1. We show how to deal with the existential strategy quantifier (the universal quantifier is symmetric). Apply Lemma 1 and reduce $(S, s) \models_{mem} \langle\!\langle A \rangle\!\rangle \psi$ to $(S_\psi, s) \models_{mem} \langle\!\langle A \rangle\!\rangle lin(\psi)$. To solve the latter, observe the following: i) each agent has finitely many observational memoryless strategies (each of polynomial size) in S , ii) instantiating a set Σ_A of such strategies, one for each agent in A , results in a sub-area S' of S_ψ in which we have to check whether $(S', s) \models A lin(\psi)$. By Proposition 1, each step ii) can be done in PSPACE. Thus one can, in PSPACE, search for a set of observational memoryless strategies Σ_A such that ii) holds. \square

Existential Fragment of **Prompt-KATL**^{*} with co-operative strategies

In this section we prove that model checking the existential fragment of **Prompt-KATL**^{*} with co-operative strategies is 2EXPTIME-complete. We begin with some definitions.

Define \models_{co}^k like \models^k except replace the definition of the semantics of the strategy quantifiers $\langle\!\langle A \rangle\!\rangle, [[A]]$ as follows:

- $(S, s) \models_{co}^k \langle\!\langle A \rangle\!\rangle \psi$ (resp. $[[A]]\psi$) iff there exists a set (resp. for all sets) Σ_A of strategies, one strategy for each agent in A , that is co-operatively observational, such that for all computations (resp. for at least one computation) $\pi \in out(s, \Sigma_A)$ we have $(S, \pi) \models^k \psi$.

⁸One can also define a “uniform” semantics in which also the agents not quantified over (i.e. the ones not in A) are limited to memoryless strategies. Our techniques can be easily adapted also to this uniform semantics.

Define $(S, s) \models_{co} \varphi$ iff there exists $k \in \mathbb{N}$ such that $(S, s) \models_{co}^k \varphi$.

Theorem 4. *The model-checking problem for the existential fragment of Prompt-KATL* (resp. Prompt-KATL) with cooperative strategies is 2EXPTIME-complete (resp. in PTIME).*

The lower bound holds already for the existential fragment of ATL* (see the proof of Theorem 5.6 in (Alur, Henzinger, and Kupferman 2002)). For the upper bound, the proof proceeds as in previous constructions, using the marking algorithm from the proof of Theorem 1. The interesting case is the strategy quantifier $\langle\langle A \rangle\rangle$, which is dealt with by the following proposition.

Proposition 4. *The following problem is decidable: given a Prompt-LTL formula ψ , a finite iCGS S , a set of agents $A \subseteq Ag$, and state $s \in S$, decide whether $(S, s) \models_{co} \langle\langle A \rangle\rangle \psi$. Moreover, this can be done in 2EXPTIME in the number of subformulas of ψ , and polynomial time in the size of S .*

Proof. The outline of the proof is as follows: we build a two-player arena G such that (\star) : $(S, s) \models_{co} \langle\langle A \rangle\rangle \psi$ if and only if there exists $k \in \mathbb{N}$ such that Player 0 has a strategy in G that enforces the LTL goal ψ_k from s (as in Proposition 3, ψ_k is formed from ψ by replacing every subformula of the form $F_P \phi$ by $\bigvee_{i \leq k} X^i \phi$). The idea is that Player 0 corresponds to the coalition A and Player 1 to the coalition $Ag \setminus A$. We are justified in treating all players in A as a single player by our assumption of cooperative strategies for them (the antagonists in $Ag \setminus A$ are always treated as a single player since all of their strategies have to be defeated).

We then modify G to obtain a new arena G' , which allows Player 0 to choose colours (*red* or *not red*) at every second step. We then prove $(\star\star)$ for every $k \in \mathbb{N}$, if Player 0 has an observational strategy in G that enforces goal ψ_k from s then it has an observational strategy in G' that enforces, from s , the following goal (\dagger) : $col(\psi)$ holds and no colour consecutively repeats more than k times⁹, and vice versa (but with ψ_{2k} substituted for ψ_k , i.e., that if Player 0 has an observational strategy in G' that enforces (\dagger) then it has an observational strategy in G that enforces ψ_{2k}). Finally, using the fact that LTL games of imperfect information admit finite-state strategies, we will show how to decide (in 2EXPTIME) whether there exists $k \in \mathbb{N}$ such that Agent 0 has a strategy in G' that enforces (\dagger) from s . Here are some more details.

If $S = \langle Ag, AP, Act, S, \lambda, \delta, \{\sim_a : a \in Ag\} \rangle$, define G to be the iCGS $\langle \{0, 1\}, AP, Act_0 \cup Act_1, S, \lambda, \delta_G, \{\sim_0, \sim_1\} \rangle$ as follows:

- action sets $Act_0 := Act^A$ and $Act_1 := Act^{Ag \setminus A}$,
- state set S , labeling λ ,
- transition function $\delta_G : S \times Act_0 \times Act_1 \rightarrow S$ that maps $(s, (d_1, d_2)) \mapsto \delta(s, d_1 \otimes d_2)$ where $d_1 \otimes d_2 \in Act^{Ag}$ maps a to $d_1(a)$ for $a \in A$ and otherwise (for $a \in Ag \setminus A$) to $d_2(a)$,
- relation \sim_0, \sim_1 defined as follows: $s \sim_0 r$ if $s \sim_A r$, and $s \sim_1 r$ if $s \sim_{Ag \setminus A} r$.

⁹W remind the reader that $col(\psi)$ is defined before Lemma 3.

It is immediate from the definitions (of G and \models_{co}) that (\star) holds. Define G' to be the iCGS obtained from G by adding a new atom *red* and splitting every transition from s to s' using decision d , into a diamond shape, where the first decision is either to go left from s (decision d with *red*) or go right (decision d with *not red*) to one of two intermediate nodes. The choice of colour is made entirely by Player 0. Then, from each of these intermediate nodes every decision leads to the node s' . The observation sets are defined such that the intermediate nodes that are successors of indistinguishable nodes are themselves indistinguishable (and thus no new information leaks). Formally, G' is the iCGS¹⁰ $\langle \{0, 1\}, AP \cup \{\text{red}\}, Act', S', \lambda', \delta', \{\sim'_0, \sim'_1\} \rangle$ where:

- $Act'_0 := Act_0 \times \{\text{red}, \text{not red}\}$ and $Act'_1 := Act_1$,
- $S' := S \cup (S \times Act^{Ag} \times \{\text{red}, \text{not red}\})$,
- the labeling λ' maps $s \mapsto \lambda(s)$ and $(s, d, x) \mapsto \{x\}$,
- transition function $\delta' : S' \times Act'_0 \times Act'_1 \rightarrow S'$ that, for actions $(d_0, x) \in Act'_0$ and $d_1 \in Act_1$, maps state $s \in S$ and actions $((d_0, x), d_1)$ to $(s, d_0 \otimes d_1, x)$; and that maps state (s, d, x) and all actions of the players to $\delta_G(s, d)$;
- REMOVED TEXT THAT DOESNT COMPILE
- observation sets \sim'_0, \sim'_1 defined as follows: $s \sim'_i r$ if $s \sim_i r$, and $(s, d, x) \sim'_i (r, d', x')$ if $s \sim_i r$.

We now describe, for every $k \in \mathbb{N}$, the natural transformation of strategies for Player 0 between G and G' . For $h' \in \text{hist}(S')$ define $\text{proj}(h') \in \text{hist}(S)$ to be the string in which every second symbol of h' is removed. Fix an action $\alpha \in Act'_0$. An observational strategy $\sigma_0 : \text{hist}(S) \rightarrow Act_0$ in G induces the strategy $\sigma'_0 : \text{hist}(S') \rightarrow Act'_0$ in G' defined as follows. If h' ends in an element of S then $\sigma'_0(h') := (\sigma_0(\text{proj}(h')), x)$, where x is *red* if the integer part of $\frac{|\text{proj}(h')|}{k}$ is odd, and otherwise (if it is even), x is *not red* (in words, use σ_0 and swap the colour every k steps in G). If h' does not end in an element of S then define $\sigma'_0(h') := \alpha$ (recall that all transitions from the intermediate nodes go to the same destination). It is easy to see that σ'_0 is observational since σ_0 is.

Before we prove the converse, we state a useful fact that follows from an induction on the length of histories: if $\text{proj}(h'_1) \sim_0 \text{proj}(h'_2)$ then $h'_1 \sim'_0 h'_2$. Now, an observational strategy $\sigma'_0 : \text{hist}(S') \rightarrow Act'_0$ in G' induces the strategy $\sigma_0 : \text{hist}(S) \rightarrow Act_0$ in G defined as follows: $\sigma_0(h) := \sigma'_0(h')$ where $\text{proj}(h') = h$. This is well defined because, if $\text{proj}(h'_1) = \text{proj}(h'_2) = h$ then $h'_1 \sim'_0 h'_2$ (by the useful fact), and so $\sigma'_0(h'_1) = \sigma'_0(h'_2)$ (since σ'_0 is observational). To see that σ_0 is observational let $h_1 \sim_0 h_2$ be equivalent histories in G , and suppose $\text{proj}(h'_i) = h_i$. Then by the useful fact, also $h'_1 \sim'_0 h'_2$, and thus, since σ'_0 is observational, we have $\sigma'_0(h'_1) = \sigma'_0(h'_2)$.

We now establish $(\star\star)$. Fix k . A strategy σ_0 in G that ensures ψ_k implies (by Lemma 3) that every play consistent with σ_0 can be k -coloured by blocks of size exactly k in a way that satisfies $col(\psi)$. Thus, the transformed strategy σ'_0 of G' ensures (\dagger) . Conversely, suppose strategy σ'_0 in G'

¹⁰We assume that the two agents use different sets of actions Act'_0 and Act'_1 — see discussion after the definition of iCGS.

ensures (\dagger) . Then (by Lemma 3) the transformed strategy σ_0 of G ensures ψ_{2k} .

Finally, we show that Player 0 can enforce (\dagger) from s in G' if and only if Player 0 can enforce $\text{col}(\psi)$ from s in G' . The “only if” direction is immediate from the definitions. For the “if” direction, it is implicit in (Kupferman and Vardi 1997a; Kupferman and Vardi 1997b) that LTL games of imperfect information admit finite-state strategies. We claim that if a strategy uses memory k then it $(|S| \times k)$ -colours the play. Indeed, let h be a history, and take $i < j \leq |h|$ such that the infix between positions i and j is of length at least $|S| \times k$. Hence, there exists $i \leq i' < j' \leq j$ such that the last states of $x := h_{\leq i'}$ and $y := h_{\leq j'}$ are equal, and the strategy is in the same memory-state after processing the histories x and y . In particular, the strategy gives the same action on x as on y . Thus, the opponent can repeatedly play the infix between i' and j' , and thus this infix must contain a colour change since we assumed that the strategy enforces $\text{col}(\psi)$ (which states, in particular, that the colours alternate infinitely often). This completes the “if” direction. \square

Conclusion

Reasoning about promptness has recently received attention for linear specifications (Alur et al. 2001; Kupferman, Piterman, and Vardi 2009; Zimmermann 2013). This is due to the fact that questions like “a specific state is eventually reached in a computation” have a clear meaning and application in the theory of formal verification, but are useless in practical scenarios if there is no bound on the time before the specific state is reached.

In this work, we initiated the study of prompt requirements over branching time specifications for multi-agent systems. We introduced the logic **Prompt-KATL***, an extension of the classic **ATL***, in which we added the prompt eventuality temporal operator F_P , and settled the model-checking complexity of many natural fragments, under various restrictions, i.e., perfect information, memoryless strategies, and the existential fragment with co-operative strategies. In particular, we prove that model-checking **Prompt-KCTL*** is PSPACE-complete without any restrictions. Note that in the case of two players the restriction to co-operative strategies is not a real restriction as these correspond to ordinary strategies. Our results for **Prompt-KATL*** and **Prompt-KATL** are summarised in the following tables:

Perfect Info. or co-operative \exists Fragment	Memoryless
2EXPTIME-complete	PSPACE-complete

Figure 1: Complexity of model checking **Prompt-KATL***

Perfect Info. or co-operative \exists Fragment	Memoryless
in PTIME	in PSPACE

Figure 2: Complexity of model checking **Prompt-KATL**

As our results show, the complexity of model checking the considered logics with the prompt operator is the same

as without the prompt operator ¹¹.

We remark that our upper-bounds for **Prompt-KATL** were obtained essentially as a side-effect of the results for **Prompt-KATL***, i.e., by analysing the complexity of the algorithms we provided for **Prompt-KATL*** in the restricted case of **Prompt-KATL**. However, one can directly reason on **Prompt-KATL**. For example, in the perfect-information case, one can show that (like for **Prompt-CTL**) $S \models \varphi$, for a **Prompt-ATL** formula φ , iff $S \models \phi$, where ϕ is the ATL formula obtained by replacing every prompt-eventuality F_P in φ by a regular eventuality F . The underlying reason is that in **Prompt-KATL** F_P only ranges over state sub-formulas and thus, reasoning about sub-formulas of the form $\langle\langle A \rangle\rangle F_P \psi$ and $[A] F_P \psi$ reduces to reasoning about two player games with reachability goals. In such games a (memoryless¹²) winning strategy does not admit a lasso in which ψ does not hold on all its states (since then the opponent can pump the loop of the lasso and win). Thus, if the goal ψ is achieved, it is achieved within a number of steps that is at most the size of the model, and thus promptly. For the case of imperfect information, more complicated reasoning can be employed.

Our work opens up several avenues of research. First, an intriguing open question is whether model-checking **Prompt-KATL*** under cooperative strategies is decidable, even for two players (in this work we established the optimal complexity for its existential fragment). Second, the satisfiability problem has yet to be tackled, and we believe that the techniques introduced in this paper would yield useful for this investigation.

Acknowledgments. Benjamin Aminof and Florian Zuleger are supported by the Austrian National Research Network S11403-N23 (RiSE) of the Austrian Science Fund (FWF) and by the Vienna Science and Technology Fund (WWTF) through grant ICT12-059. Sasha Rubin is a Marie Curie fellow of the Istituto Nazionale di Alta Matematica. Aniello Murano was supported in part by GNCS 2016 project: Logica, Automi e Giochi per Sistemi Auto-adattivi.

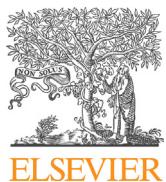
References

- [Almagor, Hirshfeld, and Kupferman 2010] Almagor, S.; Hirshfeld, Y.; and Kupferman, O. 2010. Promptness in omega-regular automata. In *ATVA 2010*, LNCS 6252, 22–36. Springer.
- [Alur et al. 2001] Alur, R.; Etessami, K.; La Torre, S.; and Peled, D. 2001. Parametric temporal logic for model measuring. *ACM Transactions on Computational Logic* 2(3):388–407.
- [Alur, Henzinger, and Kupferman 2002] Alur, R.; Henzinger, T.; and Kupferman, O. 2002. Alternating-Time Temporal Logic. *Journal of the ACM* 49(5):672–713.
- [Aminof et al. 2013] Aminof, B.; Legay, A.; Murano, A.; Serre, O.; and Vardi, M. Y. 2013. Pushdown module checking with imperfect information. *Inf. Comput.* 223:1–17.

¹¹except for the case of **Prompt-KATL** under observational memoryless strategies.

¹²Winning strategies in such games can be taken w.l.o.g. to be memoryless.

- [Aminof, Murano, and Vardi 2007] Aminof, B.; Murano, A.; and Vardi, M. Y. 2007. Pushdown module checking with imperfect information. In *CONCUR 2007*, 460–475.
- [Bulling and Jamroga 2014] Bulling, N., and Jamroga, W. 2014. Comparing variants of strategic ability: how uncertainty and memory influence general properties of games. *Journal of Autonomous Agents and Multi-Agent Systems* 28(3):474–518.
- [Čermák et al. 2014] Čermák, P.; Lomuscio, A.; Mogavero, F.; and Murano, A. 2014. MCMAS-SLK: A Model Checker for the Verification of Strategy Logic Specifications. In *CAV’14*, LNCS 8559, 524–531. Springer.
- [Čermák, Lomuscio, and Murano 2015] Čermák, P.; Lomuscio, A.; and Murano, A. 2015. Verifying and Synthesising Multi-Agent Systems against One-Goal Strategy Logic Specifications. In *AAAI 2015*, 2038–2044. AAAI Press.
- [Chatterjee, Henzinger, and Horn 2009] Chatterjee, K.; Henzinger, T. A.; and Horn, F. 2009. Finitary winning in ω -regular games. *ACM Transactions on Computational Logic* 11(1):1.
- [Chatterjee, Henzinger, and Piterman 2010] Chatterjee, K.; Henzinger, T.; and Piterman, N. 2010. Strategy Logic. *Information and Computation* 208(6):677–693.
- [Clarke and Emerson 1981] Clarke, E., and Emerson, E. 1981. Design and Synthesis of Synchronization Skeletons Using Branching-Time Temporal Logic. In *LP’81*, LNCS 131, 52–71. Springer.
- [Dima and Tiplea 2011] Dima, C., and Tiplea, F. 2011. Model-checking ATL under Imperfect Information and Perfect Recall Semantics is Undecidable. Technical report, arXiv.
- [Emerson and Halpern 1986] Emerson, E., and Halpern, J. 1986. “Sometimes” and “Not Never” Revisited: On Branching Versus Linear Time. *Journal of the ACM* 33(1):151–178.
- [Fijalkow and Zimmermann 2012] Fijalkow, N., and Zimmermann, M. 2012. Cost-parity and cost-streett games. In *FSTTCS*, volume LIPIcs 18, 124–135.
- [Huang and van der Meyden 2014] Huang, X., and van der Meyden, R. 2014. An epistemic strategy logic. *arXiv preprint arXiv:1409.2193*.
- [Huang, Luo, and Van Der Meyden 2011] Huang, X.; Luo, C.; and Van Der Meyden, R. 2011. Improved bounded model checking for a fair branching-time temporal epistemic logic. In *MoChArt*. Springer. 95–111.
- [Jamroga and Ågotnes 2007] Jamroga, W., and Ågotnes, T. 2007. Constructive knowledge: what agents can achieve under imperfect information. *J. Applied Non-Classical Logics* 17(4):423–475.
- [Jamroga and Murano 2015] Jamroga, W., and Murano, A. 2015. Module checking of strategic ability. In *AAMAS 2015*, 227–235.
- [Kupferman and Vardi 1997a] Kupferman, O., and Vardi, M. Y. 1997a. Module checking revisited. In *CAV’97*, 36–47. Springer.
- [Kupferman and Vardi 1997b] Kupferman, O., and Vardi, M. 1997b. Synthesis with incomplete information. In *2nd International Conference on Temporal Logic*, 91–106.
- [Kupferman, Piterman, and Vardi 2009] Kupferman, O.; Piterman, N.; and Vardi, M. Y. 2009. From liveness to promptness. *Formal Methods in System Design* 34(2):83–103.
- [Kupferman, Vardi, and Wolper 2000] Kupferman, O.; Vardi, M.; and Wolper, P. 2000. An Automata Theoretic Approach to Branching-Time Model Checking. *Journal of ACM* 47(2):312–360.
- [Lomuscio, Qu, and Raimondi 2009] Lomuscio, A.; Qu, H.; and Raimondi, F. 2009. MCMAS: A Model Checker for the Verification of Multi-Agent Systems. In *CAV’09*, LNCS 5643, 682–688. Springer.
- [Mogavero et al. 2014] Mogavero, F.; Murano, A.; Perelli, G.; and Vardi, M. 2014. Reasoning About Strategies: On the Model-Checking Problem. *ACM Transactions on Computational Logic* 15(4):34:1–42.
- [Mogavero, Murano, and Sorrentino 2013] Mogavero, F.; Murano, A.; and Sorrentino, L. 2013. On Promptness in Parity Games. In *LPAR’13*, 601–618. Springer.
- [Pnueli and Rosner 1989] Pnueli, A., and Rosner, R. 1989. On the Synthesis of a Reactive Module. In *POPL’89*, 179–190. Association for Computing Machinery.
- [Queille and Sifakis 1981] Queille, J., and Sifakis, J. 1981. Specification and Verification of Concurrent Programs in Cesar. In *International Symposium on Programming’81*, LNCS 137, 337–351. Springer.
- [Reif 1984] Reif, J. H. 1984. The complexity of two-player games of incomplete information. *J. Comput. Syst. Sci.* 29(2):274–301.
- [Sistla and Clarke 1985] Sistla, A. P., and Clarke, E. M. 1985. The complexity of propositional linear temporal logics. *J. ACM* 32(3):733–749.
- [van der Hoek and Wooldridge 2002] van der Hoek, W., and Wooldridge, M. 2002. Tractable Multiagent Planning for Epistemic Goals. In *Autonomous Agents and Multiagent Systems’02*, 1167–1174.
- [Vardi and Wolper 1994] Vardi, M. Y., and Wolper, P. 1994. Reasoning about infinite computations. *Information and Computation* 115(1):1–37.
- [Vester 2013] Vester, S. 2013. Alternating-time temporal logic with finite-memory strategies. In *GandALF 2013*, 194–207.
- [Zimmermann 2013] Zimmermann, M. 2013. Optimal bounds in parametric LTL games. *Theor. Comput. Sci.* 493:30–45.



Contents lists available at ScienceDirect

Information and Computation

www.elsevier.com/locate/yincoFirst-cycle games [☆]Benjamin Aminof^a, Sasha Rubin^{b,*1}^a Technische Universität Wien, Austria^b Università degli Studi di Napoli "Federico II", Italy

ARTICLE INFO

Article history:

Received 21 November 2014

Available online xxxx

Keywords:

Graph games

Cycle games

Memoryless determinacy

Parity games

Mean-payoff games

Energy games

ABSTRACT

First-cycle games (FCG) are played on a finite graph by two players who push a token along the edges until a vertex is repeated, and a simple cycle is formed. The winner is determined by some fixed property Y of the sequence of labels of the edges (or nodes) forming this cycle. These games are intimately connected with classic infinite-duration games such as parity and mean-payoff games. We initiate the study of FCGs in their own right, as well as formalise and investigate the connection between FCGs and certain infinite-duration games.

We establish that (for efficiently computable Y) the problem of solving FCGs is PSPACE-complete; we show that the memory required to win FCGs is, in general, $\Theta(n)!$ (where n is the number of nodes in the graph); and we give a full characterisation of those properties Y for which all FCGs are memoryless determined.

We formalise the connection between FCGs and certain infinite-duration games and prove that strategies transfer between them. Using the machinery of FCGs, we provide a recipe that can be used to very easily deduce that many infinite-duration games, e.g., mean-payoff, parity, and energy games, are memoryless determined.

© 2016 Elsevier Inc. All rights reserved.

1. Introduction

Infinite-duration games are studied in computer science in the context of decidability of logical theories as well as verification and synthesis in formal methods. In particular, reactive systems, which consist of an ongoing interaction between a program and its environment, may be formalised as a game played on a graph, by two players, who push a token along the edges of the graph, resulting in an infinite path, called a play. The winner of the play is determined by some winning condition. For example, in (one version of) mean-payoff games each edge in the graph carries a real number, called a weight, and Player 0 wins the play if and only if the limit-supremum of the running averages of the weights is positive; and otherwise Player 1 wins. Other types of games from the formal verification literature are reachability games, Büchi games, parity games, Muller games, energy games, etc.

Intuitively, certain games on finite graphs can be won using greedy reasoning. For instance, to win a mean-payoff game, it is sufficient for Player 0 to ensure that the average weight of each cycle that is formed is positive. This, in turn, can be ensured by enforcing the average weight of the *first cycle* formed to be positive (because Player 0 could then "forget" that the cycle was formed, and play another cycle with positive average weight, and so on). Thus, in some cases, one can

[☆] Some of the results in this paper were reported in the Proceedings of the Second International Workshop on Strategic Reasoning (SR), April 2014.

* Corresponding author.

E-mail addresses: benj@forsyte.at (B. Aminof), rubin@unina.it (S. Rubin).

¹ Marie Curie fellow of the Istituto Nazionale di Alta Matematica.

reduce reasoning about infinite-duration games to *first-cycle games*, i.e., games in which the winner is determined by the first cycle on the play. Such reasoning appears in [6], where first-cycle mean-payoff games were defined and used to prove that mean-payoff games can be won using memoryless strategies (i.e., the next move of a player does not depend on the full history up till now, but only on the current node of the play). Memoryless strategies are extremely useful, e.g., they are used to prove deep results in the theory of automata (e.g., Rabin's theorem that automata on infinite trees can be complemented), and to prove upper bounds on the complexity of solving certain classes of games (e.g., that solving parity games is in $\text{NP} \cap \text{co-NP}$).

In this paper we study first-cycle games in their own right and formalise the greedy reasoning above which connects first-cycle games with certain infinite-duration games. We now discuss our main contributions.

First-cycle games. We define first-cycle games (FCGs). These games are played on a finite graph (called an arena) by two players who push a token along the edges of the graph until a cycle is formed. Player 0 wins the play if the sequence of labels of the edges (or nodes) of the cycle is in some fixed set Y , and otherwise Player 1 wins. The set Y is called a cycle property, and we say that a cycle satisfies Y if its sequence of labels is in Y . For example, if every vertex is labelled by an integer priority, and $Y = \text{cyc-Parity}$ comprises sequences whose largest priority is even, then a cycle satisfies cyc-Parity if the largest priority occurring on the cycle is even. For a fixed cycle property Y , we write $\text{FCG}(Y)$ for the family of games over all possible arenas with this winning condition.

Complexity and memory requirements. We give a simple example showing that first cycle games (FCGs) are not necessarily memoryless determined. We then show that, for a graph with n nodes, whereas no winning strategy needs more than $n!$ memory (since this is enough to remember the whole history of the game), some winning strategies require at least $(\frac{n-1}{3})!$ memory (Proposition 1, Page 7). We analyse the complexity of solving FCGs and show that it is PSPACE-complete. More specifically, we show that if one can decide in PSPACE whether a given cycle satisfies the property Y , then solving the games in $\text{FCG}(Y)$ is in PSPACE; and that there is a trivially computable cycle property Y for which solving the games in $\text{FCG}(Y)$ is PSPACE-hard (Theorem 1, Page 8).

First-cycle games and infinite-duration games (Section 5). The central object that connects cycle games with infinite-duration games is the *cycles-decomposition* of a path (used for example by [12] to derive the value of a mean-payoff game). Informally, a path is decomposed by pushing the edges of the path onto a stack and, as soon as a cycle is detected in the stack, the cycle is output, popped, and the algorithm continues. This decomposes all but finitely many edges of the path into a sequence of simple cycles.

In order to connect FCGs with infinite-duration games we define the following notion: a winning condition W (such as the parity winning condition) is Y -greedy on arena \mathcal{A} if, in the game on arena \mathcal{A} with winning condition W , Player 0 is guaranteed to win by ensuring that every cycle in the cycles-decomposition of the play satisfies Y , and Player 1 is guaranteed to win by ensuring that every cycle in the cycles-decomposition does not satisfy Y (Definition 5, Page 14). We prove a *Strategy Transfer Theorem*: if W is Y -greedy on \mathcal{A} then the winning regions in the following two games on arena \mathcal{A} coincide, and memoryless winning strategies transfer between them: the infinite-duration game with winning condition W , and the FCG with cycle property Y (Theorem 7, Page 15).

To illustrate the usefulness of being Y -greedy, we instantiate the definition to well-studied infinite-duration games such as parity, mean-payoff, and energy games.

Memoryless determinacy of first-cycle games. We next address the fundamental question: for which cycle properties Y is every game in $\text{FCG}(Y)$ memoryless determined (i.e., no matter the arena)? We provide sufficient and necessary conditions for all games in $\text{FCG}(Y)$ to be memoryless determined (Theorem 6, Page 14). Although applying this characterisation may not be hard, it involves reasoning about arenas, which is sometimes inconvenient. Therefore, we also provide the following easy-to-check conditions on Y that ensure memoryless determinacy for all games in $\text{FCG}(Y)$ (Theorem 9, Page 16): Y is closed under cyclic permutations (i.e., if $ab \in Y$ then $ba \in Y$), and both Y and its complement are closed under concatenation (a set of strings X is closed under concatenation if $a, b \in X$ implies $ab \in X$). We demonstrate the usefulness of these conditions by observing that natural cycle properties are easily seen to satisfy them, e.g., cyc-Parity, cyc-MeanPayoff_v (which states that the limsup average of the weights is at most v), and cyc-Energy (which states that the sum of the weights is positive).

Easy-to-follow recipe. We integrate the previous results by providing an easy-to-follow recipe that allows one to deduce memoryless determinacy of all classic games that are memoryless determined and many others besides (Section 8). The recipe states that, given a winning condition W , first “finitise” W to get a cycle property Y that is closed under cyclic permutations, then prove that W is Y -greedy on the arenas of interest (usually all arenas), and finally, either show that both Y and its complement are closed under concatenation, or show that W is prefix-independent. For example, if W is the parity winning condition (i.e., the largest priority occurring infinitely often is even), the natural “finitisation” of W is the cycle property $Y = \text{cyc-Parity}$ (i.e., sequences of priorities whose largest priority is even), and it is almost completely trivial to apply the recipe in this case.

Related work. The relationship of our work with that in [6] is as follows. First, our paper deals with qualitative games (i.e., a play is either won or lost) whereas [6] consider quantitative (i.e., a play is assigned a real number, which Player 0 wants to minimize and Player 1 wants to maximize) mean-payoff games. Their main result states that mean-payoff games

are memoryless determined. However, they simultaneously prove that first-cycle games with $Y = \text{cyc-MeanPayoff}_v$ are memoryless determined. We generalise (in the qualitative setting) both of these facts and their proofs.

Referring to their proofs, [6, Page 111] state: “Our proofs are roundabout, we use the infinite game F to establish facts about the finite game G and vice versa. Perhaps more direct proofs would be desirable.” In contrast, the proof of the characterisation of memoryless determined FCGs (Theorems 4, 5 and 6) is direct, and does not go through infinite-duration games. Nonetheless, we use their idea of inducting on the choice nodes of the players, and of employing “reset” nodes in the arena.

We briefly discuss other related work. We point out (in Theorem 3, Page 10) that first-cycle games are not necessarily memoryless determined, even if the cycle property Y is closed under cyclic permutations contrary to the claim in [4, Page 370]. Our work thus also supplies a correct proof Lemma 4 in [5] which relied on this incorrect claim (see Remark 2, Page 16). Conditions that ensure (or characterise) which winning conditions always admit memoryless strategies appear in [3,8,10]. However, none of these exploit the connection to first-cycle games. For example, [8] give a full characterization of winning conditions for which all infinite-duration games are memoryless determined. Unlike our framework, the main objects of study in their work are winning conditions (i.e., languages of infinite sequences of labels) and one cannot use their results to reason about cycles in an arena since every cycle in an arena induces a cycle in the sequences of labels, but not vice versa. Their characterisation of those winning conditions which admit memoryless determined games is very “infinite” in nature, as it involves reasoning about preference relations among infinite words, and their properties concerning all infinite subsets of words recognizable by nondeterministic automata. On the other hand, their result, that if all solitaire games with a given winning condition are memoryless determined then so are all the two player games, provides a much more useful tool.

The present paper differs from the preliminary version of this work [1] mainly in the following respects: we have re-organised some of the definitions, supplied all proofs, established a full characterisation of those cycle properties Y such that every FCG over Y and Y -greedy games are memoryless determined, and extended the easy-to-follow recipe to include the case that the winning condition is prefix-independent.

2. Games

In this paper all games are two-player turn-based games of perfect information played on finite graphs. The players are called Player 0 and Player 1.

Arenas. An *arena* is a labelled directed graph $\mathcal{A} = (V_0, V_1, E, \mathbb{U}, \lambda)$ where

1. V_0 and V_1 are disjoint sets of *vertices* (alternatively, *nodes*) of Player 0 and Player 1, respectively; the set of vertices of the arena is $V := V_0 \cup V_1$, and is assumed to be finite and non-empty.
2. $E \subseteq V \times V$ is a set of *edges* with no dead-ends (i.e., for every $v \in V$ there is some edge $(v, w) \in E$).
3. \mathbb{U} is a set of possible *labels* (typical choices for \mathbb{U} are \mathbb{Q} and \mathbb{N}).
4. $\lambda : E \rightarrow \mathbb{U}$ is a *labelling function* (which will be used by the winning condition).

A *sub-arena* of \mathcal{A} is an arena of the form $(V_0, V_1, E', \mathbb{U}, \lambda')$ where $E' \subseteq E$ and $\lambda'(e) = \lambda(e)$ for $e \in E'$, i.e., it may have fewer edges. If $V_0 = \emptyset$ or $V_1 = \emptyset$ then \mathcal{A} is called a *solitaire arena*.

Remark 1. Note that all the results in this paper also hold for vertex labelled arenas, by labelling every edge (v, w) by the label of its source v . In particular, we use vertex labelling in some of our examples, with this transformation in mind. Also note that transforming a vertex labelled arena to an edge-labelled one can be done by inserting an intermediate node in the middle of every edge, and giving it the edge label. However, since this transformation changes the structure of the arena, some properties that hold for vertex labelled arenas do not always transfer to edge-labelled ones.

Sequences. Let \mathbb{N} denote the positive integers. The i th element (for $i \in \mathbb{N}$) in a sequence u is denoted u_i . The *length* of u , denoted $|u|$, is the cardinality of the sequence u . For $1 \leq i \leq j \leq |u|$, write $u[i, j]$ for the sub-sequence $u_i u_{i+1} \cdots u_j$. The empty sequence is denoted ϵ . By convention, $u[i, j] = \epsilon$ if $j < i$. The set of infinite sequences over alphabet X is written X^ω , the set of finite sequences is written X^* . We write concatenation as $u \cdot v$ or simply as uv .

Paths, simple paths, and node-paths. For an edge $e = (v, w)$ we call v the *start* and w the *end* of e , and write $\text{start}(e) := v$ and $\text{end}(e) := w$; we say that v and w *appear on the edge* e . A *path* π in \mathcal{A} is a finite or infinite sequence of edges $\pi_1 \pi_2 \cdots \in E^* \cup E^\omega$ such that $\text{end}(\pi_i) = \text{start}(\pi_{i+1})$ for $1 \leq i < |\pi|$. The set of paths in \mathcal{A} is denoted $\text{paths}(\mathcal{A})$. We extend start and end to paths in the natural way: $\text{start}(\pi) := \text{start}(\pi_1)$, and if π is of length $\ell \in \mathbb{N}$ then $\text{end}(\pi) := \text{end}(\pi_\ell)$. We say that vertex v *appears on the path* π if v appears on some edge π_i of π . We extend λ from edges to paths point-wise: $\lambda(\pi)$ is defined to be the string of labels $\lambda(\pi_1)\lambda(\pi_2)\cdots$. A path π is called *simple* if $\text{start}(\pi_i) = \text{end}(\pi_j)$ implies $i = j + 1$.

A sequence of nodes $v \in V^* \cup V^\omega$ is called a *node-path* if $(v_i, v_{i+1}) \in E$ for all i . If $\pi \in E^* \cup E^\omega$ is a path then $\text{nodes}(\pi)$ is a node-path, where $\text{nodes}(\pi)$ is defined to be the sequence $\text{start}(\pi_1)\text{start}(\pi_2)\cdots$ if π is infinite, and $\text{start}(\pi_1)\cdots\text{start}(\pi_{|\pi|})\text{end}(\pi_{|\pi|})$ if it is finite. Every node-path is either a singleton sequence v or of the form $\text{nodes}(\pi)$

for some path π . For a finite non-empty sequence u of vertices (such as a finite node-path), we overload notation and write $\text{end}(u)$ for the last node in u .

Histories and strategies. A strategy for a player is a function that tells the player what node to move to given the history, i.e., the sequence of nodes visited so far. Define the set $H_\sigma(\mathcal{A})$ of histories for Player $\sigma \in \{0, 1\}$ by $H_\sigma(\mathcal{A}) := \{u \in V^*V_\sigma : (u_n, u_{n+1}) \in E, 1 \leq n < |u|\}$. In other words, $H_\sigma(\mathcal{A})$ is the set of node-paths ending in V_σ . A strategy for Player σ is a function $S : H_\sigma(\mathcal{A}) \rightarrow V$ such that $(\text{end}(u), S(u)) \in E$ for all $u \in H_\sigma(\mathcal{A})$. A strategy S for Player σ is *memoryless* if $S(u) = S(u')$ for all $u, u' \in H_\sigma(\mathcal{A})$ with $\text{end}(u) = \text{end}(u')$. Hence, we usually consider a memoryless strategy as a function $S : V_\sigma \rightarrow V$. A node-path $u \in V^* \cup V^\omega$ is *consistent* with a strategy S for Player σ if whenever $u_n \in V_\sigma$ (for $n < |u|$) then $u_{n+1} = S(u[1, n])$. A path π is *consistent* with a strategy S if $\text{nodes}(\pi)$ is consistent with S .

A strategy S for Player σ is *generated by a Mealy machine* (M, m_1, δ, μ) if there exists a finite set M of *memory states*, an *initial state* $m_1 \in M$, a *memory update function* $\delta : V \times M \rightarrow M$, and a *next-move function* $\mu : V_\sigma \times M \rightarrow V$, such that for a history $u = u_1 u_2 \dots u_l \in H_\sigma(\mathcal{A})$ we have $S(u) = \mu(u_l, m_l)$ where m_l is defined inductively by $m_1 = m_1$ and $m_{l+1} = \delta(u_l, m_l)$. A strategy S is *finite-memory* if it is generated by some Mealy machine. A strategy S uses memory at most k if it is generated by some Mealy machine with $|M| \leq k$, and it uses memory at least k if every Mealy machine generating S has $|M| \geq k$.

Notational abuse. We sometimes, for a path $\pi \in E^*$, write $S(\pi)$ instead of $S(\text{nodes}(\pi))$.

Plays. An infinite path in \mathcal{A} is called a *play*.² We denote the set of all plays of \mathcal{A} by $\text{plays}(\mathcal{A})$. For a node $v \in V$, and strategy S , we write $\text{plays}_\mathcal{A}(S, v)$ for the set of plays π that start with v and are consistent with S ; we write $\text{reach}_\mathcal{A}(S, v)$ for the set of vertices in V that appear on the plays in $\text{plays}_\mathcal{A}(S, v)$. We may drop the subscript \mathcal{A} when the arena is clear from the context.

Games and winning. A game is a pair (\mathcal{A}, O) consisting of an arena $\mathcal{A} = (V_0, V_1, E, \mathbb{U}, \lambda)$ and an objective $O \subseteq \text{plays}(\mathcal{A})$. Objectives are usually determined by winning conditions or cycle properties (defined later). A play π in a game (\mathcal{A}, O) is *won by Player 0* if $\pi \in O$, and otherwise it is *won by Player 1*. A strategy S for Player σ is *winning from* a node $v \in V$ (in the game (\mathcal{A}, O)) if every play in $\text{plays}_\mathcal{A}(S, v)$ is won by Player σ . If Player σ has a winning strategy from v we say that Player σ *wins from* v . A game (\mathcal{A}, O) is said to be *determined* if, for every $v \in V$, one of the players wins from v .

The *winning region* (resp. *memoryless winning region*) of Player σ is the set of vertices $v \in V$ such that Player σ has a winning strategy (resp. memoryless winning strategy) from v . We denote the winning region in the game (\mathcal{A}, O) of Player σ by $\text{WR}^\sigma(\mathcal{A}, O)$.

Solitaire games and memoryless strategies. If either V_0 or V_1 is empty, then the game (\mathcal{A}, O) is called a *solitaire game*.

A simple property of memoryless strategies that is often used is the following. In a game over an arena $\mathcal{A} = (V_0, V_1, E, \mathbb{U}, \lambda)$, every memoryless strategy S for Player σ induces a sub-arena $\mathcal{A}^{|S|} = (V_0, V_1, E^{|S|}, \mathbb{U}, \lambda^{|S|})$, obtained by deleting all edges $(v, w) \in E$ where $v \in V_\sigma$ and $w \neq S(v)$. Observe that in $\mathcal{A}^{|S|}$ all Player σ nodes have exactly one outgoing edge (namely the edge specified by S), and thus Player σ has no choices to make in this arena. For this reason, many times one considers the solitaire arena $\mathcal{A}^{\|S\|}$ in which all the nodes are assigned to Player $1 - \sigma$, i.e., $\mathcal{A}^{\|S\|} = (V_0^{\|S\|}, V_1^{\|S\|}, E^{\|S\|}, \mathbb{U}, \lambda^{\|S\|})$, where $V_\sigma^{\|S\|} = \emptyset$, and $V_{1-\sigma}^{\|S\|} = V$. The main connection between \mathcal{A} , $\mathcal{A}^{|S|}$ and $\mathcal{A}^{\|S\|}$ is that every path in $\mathcal{A}^{|S|}$ ($\mathcal{A}^{\|S\|}$) is a path in \mathcal{A} that is also consistent with S , and vice versa. We can thus reason about paths in $\mathcal{A}^{|S|}$ (or $\mathcal{A}^{\|S\|}$) instead of paths in \mathcal{A} that are consistent with S .

Memoryless determinacy. A game is *memoryless from* v if the player that wins from v has a memoryless strategy that is winning from v .

A game is *point-wise memoryless for Player σ* if the memoryless winning region for Player σ coincides with the winning region for Player σ . A game is *uniform memoryless for Player σ* if there is a memoryless strategy for Player σ that is winning from every vertex in that player's winning region.

A game is *point-wise memoryless determined* if it is determined and it is point-wise memoryless for both players. A game is *uniform memoryless determined* if it is determined and uniform memoryless for both players.

Winning conditions. A *winning condition* is a set $W \subseteq \mathbb{U}^\omega$. If W is a winning condition and \mathcal{A} is an arena, the objective $O^\mathcal{A}(W)$ induced by W is defined as follows: $O^\mathcal{A}(W) = \{\pi \in \text{plays}(\mathcal{A}) : \lambda(\pi) \in W\}$. For ease of readability we may drop the superscript \mathcal{A} and write $O(W)$ instead of $O^\mathcal{A}(W)$.

Here are some standard winning conditions:

- The *parity condition* consists of those infinite sequences $c_1 c_2 \dots \in \mathbb{N}^\omega$ such that the largest label occurring infinitely often is even.
- For $v \in \mathbb{R}$, the *v -mean-payoff condition* consists of those infinite sequences $c_1 c_2 \dots \in \mathbb{R}$ such that $\limsup_{k \rightarrow \infty} \frac{1}{k} \sum_{i=1}^k c_i$ is at most v .

² For ease of presentation, we consider plays of both finite- and infinite-duration games to be infinite. Obviously, in games finite-duration, games such as FCGs, the winner is determined by a finite prefix of the play, and the moves after this prefix are immaterial.

- The *energy condition*, for a given *initial credit* $r \in \mathbb{N}$, consists of those infinite sequences $c_1c_2\cdots \in \mathbb{Z}^\omega$ such that $r + \sum_{i=1}^k c_i \geq 0$ for all $k \geq 1$.
- The *energy-parity condition* (for a given initial credit r) is defined as consisting of $(c_1, d_1)(c_2, d_2)\cdots \in \mathbb{N} \times \mathbb{Z}$ such that $c_1c_2\cdots$ satisfies the parity condition and $d_1d_2\cdots$ satisfies the energy condition with initial credit r .

3. Cycles and games

In this section we define the cycles-decomposition of a path, as well as the first-cycle game, and the infinite-duration all-cycles game.

Cycles-decomposition. A *cycle* in an arena \mathcal{A} is a finite path π such that $\text{start}(\pi) = \text{end}(\pi)$. Note that, like paths, cycles are ordered. Hence, the cycles $(v_1, v_2)(v_2, v_1)$ and $(v_2, v_1)(v_1, v_2)$ are not identical.

Algorithm 1 CycDEC(s, π).

```

Require:  $s$  is a finite (possibly empty) simple path                                ▷ initial stack content
Require:  $\pi$  is a finite or infinite path  $\pi_1\pi_2\cdots$                                 ▷ the path to decompose
Require: If  $s$  is non-empty then  $\text{end}(s) = \text{start}(\pi)$                                 ▷  $s\pi$  must form a path

step = 1
while step ≤ |π| do
    Append  $\pi_{step}$  to  $s$                                               ▷ Start a step
    Say  $s = e_1e_2\cdots e_m$                                                ▷ Push current edge into stack
    if  $\exists i : e_j e_{j+1} \cdots e_m$  is a cycle then
        Output  $e_i e_{i+1} \cdots e_m$                                          ▷ If stack has a cycle
         $s := e_1 \cdots e_{i-1}$                                                  ▷ output the cycle
    end if
    step := step + 1                                                       ▷ Pop the cycle from the stack
end while

```

The code appearing in [Algorithm 1](#) defines an algorithm CycDEC that takes as input a (usually empty) simple path s , which is treated as the initial contents of a stack, and a path π (finite or infinite). At step $j \geq 1$, the edge π_j is pushed onto the stack and if, for some k , the top k edges on the stack form a cycle, this cycle is output, then popped, and the procedure continues to step $j + 1$.

Note that CycDEC may take a finite path π and non-empty stack s as input. Moreover, it halts if and only if π is a finite path.

The sequence of cycles output by this procedure when input the empty stack $s = \epsilon$ and path π , denoted *cycles*(π), is called the *cycles-decomposition* of π . The *first cycle* of π , written *first*(π), is the first cycle in *cycles*(π). For example, if

$$\pi = (v, w)(w, x)(x, w)(w, v)(v, s)(s, x)[(x, y)(y, z)(z, x)]^\omega,$$

then

$$\text{cycles}(\pi) = (w, x)(x, w), (v, w)(w, v), (x, y)(y, z)(z, x), (x, y)(y, z)(z, x), \dots,$$

and the first cycle of π is $(w, x)(x, w)$.

It is easy to see that, if the algorithm CycDEC is run on a path π in an arena with n nodes, then at most $n - 1$ edges of π are pushed but never popped (like the edges (v, s) and (s, x) in the example above).³ So we have:

Lemma 1. For every path π in arena \mathcal{A} with vertex set V , there are at most $|V| - 1$ indices i such that π_i does not appear in any of the cycles in *cycles*(π).

Given a play π in \mathcal{A} , an initial stack content s , and $i \geq 0$, we write $\text{stack}^i(s, \pi)$ for the contents of the stack of algorithm CycDEC(s, π) at the beginning of step $i + 1$ (i.e., just before the edge π_{i+1} is pushed). Note that if $\text{stack}^i(s, \pi)$ is not empty then it ends with the vertex $\text{start}(\pi_{i+1})$, whether or not a cycle was popped during step i . For $i \geq 1$, we define $\text{cycle}^i(s, \pi)$ to be the cycle output by the algorithm CycDEC(s, π) during step i , or ϵ if no cycle was output at step i . We may drop s when $s = \epsilon$, and we may drop i when $i = |\pi|$. For instance, $\text{stack}(\pi)$ is the stack content at the end of the algorithm CycDEC(ϵ, π) for a finite π ; and $\text{cycle}^i(\pi)$ is the cycle output during step i of the algorithm CycDEC(ϵ, π).

Given that, for every $i \geq 1$, the behaviour of the algorithm CycDEC(s, π) from the end of step i onwards is completely determined by $\text{stack}^i(s, \pi)$, and the suffix $\pi_{i+1}\pi_{i+2}\dots$ of π , the following lemma is immediate:

Lemma 2. Let π be a play in \mathcal{A} , let $i \geq 0$, let $w = \text{stack}^i(\pi)$, and let $\pi' = \pi_{i+1}\pi_{i+2}\dots$ and $\pi'' = w \cdot \pi'$. Then for every $j \geq 0$ we have that

³ As we show in Section 8, this allows one to reason, for instance, about the initial credit problem for energy games.

1. $\text{stack}^{i+j}(\pi) = \text{stack}^j(w, \pi') = \text{stack}^{|w|+j}(\pi'')$, and
2. $\text{cycle}^{i+j}(\pi) = \text{cycle}^j(w, \pi') = \text{cycle}^{|w|+j}(\pi'')$.

Furthermore, for every $0 \leq l \leq |w|$, we have that $\text{stack}^l(\pi'') = w_1 \dots w_l$, and $\text{cycle}^l(\pi'') = \epsilon$.

Cycle properties. For a given \mathbb{U} , a *cycle property* is a set $Y \subseteq \mathbb{U}^*$, used later on to define winning conditions for games.⁴ Here are some cycle properties that we refer to in the rest of the article:

1. Let cyc-EvenLen be those sequences $c_1 c_2 \dots c_k \in \mathbb{U}^*$ such that k is even.
2. Let cyc-Parity be those sequences $c_1 \dots c_k \in \mathbb{N}^*$ such that $\max_{1 \leq i \leq k} c_i$ is even.
3. Let cyc-Energy be those sequences $c_1 \dots c_k \in \mathbb{Z}^*$ such that $\sum_{i=1}^k c_i \geq 0$.
4. Let cyc-GoodForEnergy be those sequences $(c_1, d_1) \dots (c_k, d_k) \in (\mathbb{N} \times \mathbb{Z})^*$ such that $\sum_{i=1}^k d_i > 0$, or both $\sum_{i=1}^k d_i = 0$ and $c_1 \dots c_k \in \text{cyc-Parity}$.
5. Let cyc-MeanPayoff_v (for $v \in \mathbb{R}$) be those sequences $c_1 \dots c_k \in \mathbb{R}^*$ such that $\frac{1}{k} \sum_{i=1}^k c_i \leq v$.
6. Let cyc-MaxFirst be those sequences $c_1 \dots c_k \in \mathbb{N}^*$ such that $c_1 \geq c_i$ for all i with $1 \leq i \leq k$.
7. Let cyc-EndsZero be those sequences $c_1 \dots c_k \in \mathbb{N}_0^*$ such that $c_k = 0$.

If $Y \subseteq \mathbb{U}^*$ is a cycle property, write $\neg Y$ for the cycle property $\mathbb{U}^* \setminus Y$. We will usually use Y to define goals for Player 0 (hence Player 1's goals would be naturally associated with $\neg Y$). It is convenient to define $Y^0 = Y$, and $Y^1 = \neg Y$. This allows us to refer to the goals of Player σ in terms of Y^σ .

We isolate two important classes of cycle properties (the first is inspired by [4]):

1. Say that Y is *closed under cyclic permutations* if $ab \in Y$ implies $ba \in Y$, for all $a \in \mathbb{U}, b \in \mathbb{U}^*$.
2. Say that Y is *closed under concatenation* if $a \in Y$ and $b \in Y$ imply that $ab \in Y$, for all $a, b \in \mathbb{U}^*$.

Note, for example, that the cycle properties 1–5 above are closed under cyclic permutations and concatenation; and that $\neg \text{cyc-EvenLen}$ is closed under cyclic permutations but not under concatenation.

If \mathcal{A} is an arena with labelling λ , and C is a cycle in \mathcal{A} , we say that a cycle C *satisfies* Y if $\lambda(C) \in Y$.

First-cycle games and all-cycles games.

For arena $\mathcal{A} = (V_0, V_1, E, \mathbb{U}, \lambda)$ and cycle property $Y \subseteq \mathbb{U}^*$, we define two objectives (subsets of $\text{plays}(\mathcal{A})$):

1. $\pi \in O_{\text{first}}^{\mathcal{A}}(Y)$ if $\text{first}(\pi)$ satisfies Y .
2. $\pi \in O_{\text{all}}^{\mathcal{A}}(Y)$ if every cycle in $\text{cycles}(\pi)$ satisfies Y .

To ease readability, we drop the superscript \mathcal{A} when the arena is clear from the context and write, for example, $O_{\text{all}}(Y)$ instead of $O_{\text{all}}^{\mathcal{A}}(Y)$.

The game $(\mathcal{A}, O_{\text{first}}(Y))$ is called a *first-cycle game (over Y)*, and the family of all first-cycle games over Y (i.e., taking all possible arenas) is denoted $\text{FCG}(Y)$. Similarly, we write $\text{ACG}(Y)$ for the family of *all-cycles games* over Y . For instance, $\text{FCG}(\text{cyc-Parity})$ consists of those games such that Player 0 wins iff the largest label occurring on the first cycle is even.

A game (\mathcal{A}, O) is *finitary* if every play is already won after a finite number of steps, i.e., for every $\pi \in \text{plays}(\mathcal{A})$ there exists $K_\pi \in \mathbb{N}$ such that, if Player σ wins the play π , then Player σ also wins every play of \mathcal{A} starting with the prefix $\pi[1, K_\pi]$. Clearly FCGs are finitary since the winner is already determined by the first-cycle of the play (recall that arenas are finite). A basic result states that finitary games (of perfect information) are determined.⁵ We recap the proof because we use it in [Theorem 1](#) to determine the computational complexity of deciding which player has a winning strategy in a given FCG.

Lemma 3. Every finitary game (\mathcal{A}, O) is determined.

Proof. Suppose (\mathcal{A}, O) is finitary. Fix a starting node v and note that the set of finite node-paths starting in v form a tree, which we call the *game-tree*. Plays in \mathcal{A} correspond to (infinite) branches in the game-tree. Prune every branch π of the game-tree at its K_π th node to get the pruned game-tree T . By König's Tree Lemma, the pruned game-tree T is finite since it is finitely branching and contains no infinite paths. Turn T into an And-Or tree: label an internal node ρ by \vee in case $\text{end}(\rho) \in V_0$, and by \wedge in case $\text{end}(\rho) \in V_1$; label a leaf ρ by *true* if every play extending ρ is won by Player 0, and by *false* if every play extending ρ is won by Player 1. Note that every leaf gets a label. It is easy to see that Player 0 has a winning strategy from v in the game (\mathcal{A}, O) if and only if the And-Or tree evaluates to *true*. \square

⁴ When \mathbb{U} is clear from the context, we will not mention it.

⁵ This is usually attributed to Zermelo, but also follows from the fact that open games are determined, e.g., [7].

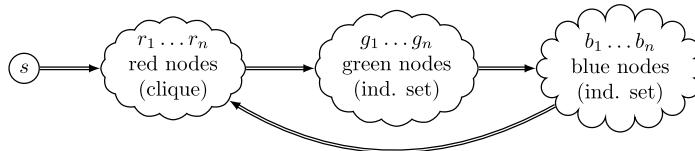


Fig. 1. double arrows represent one edge from every node to every node.

4. First-cycle games

4.1. Memory bounds and complexity

In this section we discuss the amount of memory required to win FCGs and the computational complexity of deciding which player has a winning strategy.

We begin by showing that while strategies with memory $2^{O(|V| \log |V|)}$ always suffice, there are FCGs that require $2^{\Omega(|V| \log |V|)}$ memory.

Proposition 1.

1. For a FCG on an arena with n vertices, if Player σ wins from v , then every winning strategy for Player σ starting from v uses memory at most $n!$.
2. For every $n \in \mathbb{N}$ there exists a FCG on an arena with $3n + 1$ vertices, and a vertex v , such that every winning strategy for Player 0 from v uses memory at least $n!$.

Proof. For the upper bound, note that it is enough to remember the whole history of the play until a cycle is formed, i.e., all histories of length at most $n - 1$. To store all histories of length k requires $\sum_{i \leq k} i! < (k + 1)!$ many states. Thus, $n!$ memory suffices.

We now turn to the lower bound.

Sketch. We define a game in which Player 1 can select any possible permutation of $\{1, \dots, n\}$ and, in order to win, Player 0 must remember this permutation. Consider the arena in Fig. 1. Intuitively, the red nodes are used by Player 1 to select a permutation of $\{1, \dots, n\}$, and the green and blue nodes are used by her to query Player 0 as to the relative order of any pair of indices i, j in this permutation. The outgoing edges from the blue nodes are used by Player 0 to respond to the query by going back to either r_i or r_j . Player 0 wins the game if the cycle contains both r_i and r_j , hence, he must remember the order of the pair i, j in the permutation. Since this is true for every such pair of indices, he must use at least $n!$ memory to store the permutation. The winning condition captures the above intuition, and also ensures that Player 1 loses if she deviates from the above protocol.

Details. The arena $\mathcal{A} = (V_0, V_1, E, \mathbb{U}, \lambda)$ has n red nodes r_1, \dots, r_n , n green nodes g_1, \dots, g_n , n blue nodes b_1, \dots, b_n , and an initial node s . The red nodes form a clique (i.e., $(r_i, r_j) \in E$ for every $1 \leq i \neq j \leq n$), the green nodes form an independent set, and so do the blue nodes (i.e., $(g_i, g_j) \notin E$ and $(b_i, b_j) \notin E$ for every $1 \leq i, j \leq n$). In addition, for every $1 \leq i, j \leq n$ we have the edges $(s, r_j), (r_i, g_j), (g_i, b_j), (b_i, r_j)$. Player 0 owns the blue nodes ($V_0 = \{b_1, \dots, b_n\}$), and the rest belong to Player 1. In order to correctly describe the winning condition, we label the nodes as follows: for every $1 \leq i \leq n$, the nodes r_i, g_i, b_i are labelled $(Red, i), (Green, i), (Blue, i)$, respectively; and the node s is labelled by 0. In other words, $\mathbb{U} = \{0\} \cup (\{Red, Green, Blue\} \times \{1, \dots, n\})$, and the induced edge labelling is: for every $(v, w) \in E$, we have that $\lambda(v, w) = (Red, i)$ if $v = r_i$, $\lambda(v, w) = (Green, i)$ if $v = g_i$, $\lambda(v, w) = (Blue, i)$ if $v = b_i$, and otherwise $\lambda(v, w) = 0$.

We now define the winning condition. Player 0 wins a play iff the first cycle $(v_1, v_2) \dots (v_k, v_{k+1})$ on the play satisfies one of the following three conditions⁶:

1. the node just before the end is not blue (i.e., $v_k \notin \{b_1, \dots, b_n\}$); or
2. it does not have exactly 1 green node and 1 blue node; or
3. exactly 1 green node g_j , and 1 blue node b_l appear on it, and $b_l = v_k$, and
 - (a) it has red nodes labelled with the same numbers as g_j and b_l (i.e., r_j, r_l are on the cycle), and
 - (b) it starts with a red node labelled with a number equal to that of the green or the blue node (i.e., $v_1 \in \{r_j, r_l\}$).

Informally, the first two conditions above ensure that Player 0 can win if Player 1 does not select a permutation followed by a query, whereas the third condition ensures that if Player 1 does, then Player 0 can win but only if he remembers the whole permutation.

⁶ For simplicity, we state the winning condition in terms of the nodes on the cycle and not in terms of the labels of the edges. However, since we essentially label an edge by the name of its source node, the latter can be easily done.

We first prove that Player 0 has a winning strategy.⁷ We distinguish between two types of plays:

- (i) Player 1 selects a permutation through the red nodes, and then queries Player 0 by going to some green node g_j , followed by a blue node b_l ; i.e., $u = (s, r_{i_1})(r_{i_1}, r_{i_2}) \dots (r_{i_{n-1}}, r_{i_n})(r_{i_n}, g_j)(g_j, b_l)$ are the first $n+2$ edges of the play, and for every $1 \leq x \leq n$ we have that $x = i_h$ for some $1 \leq h \leq n$. In this case, let $1 \leq \tilde{j}, \tilde{l} \leq n$ be such that $i_{\tilde{j}} = j$ and $i_{\tilde{l}} = l$, and observe that Player 0 can win by taking the edge (b_l, r_{i_h}) where $h = \min(\tilde{j}, \tilde{l})$ (i.e., by taking the edge back to the node among r_i, r_j that appears sooner on u). Indeed, the first cycle of the play will then be $(r_{i_h}, r_{i_{h+1}}) \dots (r_{i_{n-1}}, r_{i_n})(r_{i_n}, g_j)(g_j, b_l)$, and it will satisfy the third option in the winning condition (in particular, by our choice of h , both r_i and r_j are on the cycle, and $i_h \in \{j, l\}$).
- (ii) The play never reaches a blue node, or when the play reaches a blue node for the first time it does not conform to the pattern given in case (i) above. If the play never reaches a blue node then Player 0 wins since the first cycle formed satisfies the first option in the winning condition. This is also true if a cycle was formed before the play reaches a blue node for the first time. Hence, we are left with the option that the prefix of the play is of the form $u = (s, r_{i_1})(r_{i_1}, r_{i_2}) \dots (r_{i_{t-1}}, r_{i_t})(r_{i_t}, g_j)(g_j, b_l)$, where $t < n$, and for $x \neq y$ we have $r_{i_x} \neq r_{i_y}$. In this case, Player 0 first takes the edge (b_l, r_m) , where $1 \leq m \leq n$ is the index of a red node that did not yet appear on the play (i.e., $i_x \neq m$ for all $1 \leq x \leq t$). Note that this does not yet form a cycle and the play continues. Now, the game can proceed in three ways, all losing for Player 1: the first is by keeping the play forever away from blue nodes, thus closing a cycle satisfying option 1 in the winning condition; the second is by closing the first cycle by going back to b_l , again losing by option 1 (since then v_k is green); the third is by reaching some blue node b_h different from b_l without yet forming a cycle. In this last case, Player 0 wins (using the second option in the winning condition) by taking the edge (b_h, r_{i_t}) and thus forming a cycle that contains two different blue nodes: b_l, b_h .

We now show that every strategy for Player 0 that uses less than $n!$ memory is not winning. Let ξ be a Player 0 strategy that is implemented using a Mealy machine \mathcal{M} with less than $n!$ states. Hence, there are two different permutations $p = i_1, \dots, i_n$ and $p' = i'_1, \dots, i'_n$ of $\{1, \dots, n\}$, such that \mathcal{M} is in the same state m after reading the history $s \cdot r_{i_1} \dots r_{i_n}$, as after reading the history $s \cdot r_{i'_1} \dots r_{i'_n}$. Let h be the smallest index such that $i_h \neq i'_h$, and let $j := i_h$ and $l := i'_h$, and let $1 \leq \tilde{j}, \tilde{l} \leq n$ be such that $i_{\tilde{j}} = j$ and $i_{\tilde{l}} = l$. Simply put, j appears in position h in p and position \tilde{j} in p' , whereas l appears in position h in p' and position \tilde{l} in p . The minimality of h implies that $\tilde{j}, \tilde{l} > h$, and thus l appears after j in p , but before j in p' .

Observe that the two histories $\rho = s \cdot r_{i_1} \dots r_{i_n} \cdot g_j \cdot b_l$, and $\rho' = s \cdot r_{i'_1} \dots r_{i'_n} \cdot g_j \cdot b_l$ also put \mathcal{M} in the same state and thus, ξ responds to both histories with the same move, say by taking the edge (b_l, r_x) . Observe that, since p, p' are permutations, every red node already appears on ρ, ρ' . Hence, in both cases, every possible move of Player 0 from b_l closes a cycle that has exactly one green and one blue node, and the blue node appears just before the end. It follows that, in both cases, in order to win Player 0 must satisfy items 3.a and 3.b of the winning condition. In order to satisfy 3.a he must choose $r_x = r_l$ or $r_x = r_j$. However, that prevents him from satisfying item 3.b in both cases since r_j appears on ρ before r_l , but after it on ρ' . More formally, consider first the option $r_x = r_l$ and the history ρ . Recall that $i_{\tilde{l}} = l$, that $j = i_h$, and that $\tilde{l} > h$. Hence, the first cycle formed is $(r_{i_{\tilde{l}}}, r_{i_{\tilde{l}+1}}) \dots (r_{i_{n-1}}, r_{i_n})(r_{i_n}, g_j)(g_j, b_l)(b_l, r_{i_{\tilde{l}}})$, and it does not contain the required r_j . Similarly, if $r_x = r_j$ consider the history ρ' , and note that the first cycle formed is $(r_{i'_j}, r_{i'_{j+1}}) \dots (r_{i'_{n-1}}, r_{i'_n})(r_{i'_n}, g_j)(g_j, b_l)(b_l, r_{i'_j})$, and it does not contain the required r_l . It follows that ξ is not a winning strategy. \square

We now address the complexity of solving FCGs with efficiently computable cycle properties. Given a game, and a starting node v , solving the game is the problem of deciding whether Player 0 or Player 1 wins from v . We show that this problem is in general PSPACE-complete.

Theorem 1.

1. If Y is a cycle property for which solving membership is in PSPACE then the problem of solving games in $\text{FCG}(Y)$ is in PSPACE.
2. There exist a cycle property Y , that is computable in linear-time, such that the problem of solving games in $\text{FCG}(Y)$ is PSPACE-hard.

Proof. Let \mathcal{A} be an arena with n nodes. Following the proof of Lemma 3, for the special case of FCGs, prune a branch of the game-tree once the first cycle is formed, and label the game-tree to get an And-Or tree. Every node in the And-Or tree is of depth at most n , and has at most n children. Hence, evaluating the tree can be done, using a depth-first algorithm, in space polynomial in n plus the maximal space required to compute the labels of the leaves ρ in the tree. Note that the label of a leaf ρ is determined by whether $\lambda(c) \in Y$ or not (where c is the first-cycle on ρ). By our assumption on Y this can be done in polynomial space, hence the upper bound follows.

⁷ We define the strategy only for histories that are consistent with it. Obviously, it can be arbitrarily defined on any other history.

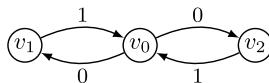


Fig. 2. Sample arena A . Circles are Player 0 nodes, solid lines are edges.

The lower bound follows immediately by reducing the PSPACE-hard problem QBF to solving $\text{FCG}(Y)$, where Y is the set of all sequences whose last two elements are identical. The proof uses the same arena (which is vertex-labelled – see Remark 1) as in the reduction of QBF to Generalised Geography (e.g., [11, Theorem 8.11]). To quickly recap, the basic idea there is to view QBF as a game in which Player 0 (resp. Player 1) assigns the values of an existentially (resp. universally) quantified variable x_i by picking a node labelled by the literal x_i or the literal $\neg x_i$; after all variables are thus assigned, Player 1 picks a clause (challenging Player 0 to show that it is true), and finally Player 0 responds by picking a literal of this clause (that he claims is true). The structure of the arena is such that the only outgoing edge from the node picked by Player 0 in this last step is the node labelled by the same literal that was available during the value-assigning phase. Thus, if indeed that literal was picked in the assignment phase, then the next move closes a cycle that satisfies Y and otherwise, taking this edge does not close a cycle, forcing Player 0 to close a losing cycle in the next step. Overall, the QBF formula is true iff Player 0 wins this FCG. \square

4.2. The relation between point-wise and uniform memoryless determinacy

In this section we consider the difference between point-wise memoryless determinacy and uniform memoryless determinacy in FCGs. We begin by considering the simple case of solitaire games.

Theorem 2. All solitaire FCGs are point-wise memoryless determined. However, some solitaire FCG's are not uniform memoryless determined.

Proof. The fact that a solitaire FCG is point-wise memoryless determined is simply because once a node repeats the game is effectively over, and thus the player makes at most one meaningful move from any node. In other words, Player σ can win from a node v iff there is a play π starting in v such that the first cycle $\pi_i \dots \pi_j$ on π satisfies Y^σ . Traversing the prefix $\pi_1 \dots \pi_j$ requires no memory since each vertex on it is the source of exactly one edge. For the second item, consider the arena in Fig. 2. Observe that, for the cycle property $Y = \text{cyc-EndsZero}$, no memoryless strategy is winning from both v_1 and v_2 . \square

We now consider two-player games. In contrast with solitaire games, some two-player FCGs are not point-wise memoryless determined. Indeed, any solitaire game (in which all nodes belong to Player 0) that is not uniform memoryless determined, can be turned into a two player game in which Player 0 wins from some node w , but requires memory to do so: simply add a single Player 1 node w that has outgoing edges to all nodes in the original game.

As we later show (Corollary 1, Page 11), if a cycle property Y is closed under cyclic permutations then all solitaire games in $\text{FCG}(Y)$ are uniform memoryless determined. Unfortunately, for two player games, this is not enough even for point-wise memoryless determinacy. Indeed, Theorem 3 shows that closure of Y under cyclic permutations is a necessary condition for having all games in $\text{FCG}(Y)$ be point-wise memoryless determined, but it is not a sufficient one.⁸

We also show that, for cycle properties Y that are closed under cyclic permutations, if a game in $\text{FCG}(Y)$ is point-wise memoryless for a player then it is also uniform memoryless for this player (Theorem 3 Item 3). The proof of this fact uses the following Lemma.

Lemma 4. Suppose Y is closed under cyclic permutations, and let S be a strategy for Player σ in arena A . If S is winning from v , then S is winning from every node $w \in \text{reach}_A(S, v)$.

Proof. We begin with the intuition.

Sketch. If c is the first cycle of some play $\pi \in \text{plays}(S, w)$, then we can construct a path π' starting in v and consistent with S whose first cycle is some cyclic permutation c' of c , as follows: we proceed along some simple path from v to w until we hit a node on π ; if this node is not on c , we continue along π until we reach a node on c ; and finally, we cycle along the nodes of c until we return to where we started, forming a cycle c' . Note that c' is not necessarily identical to c since in the first stage we may have hit π somewhere in the middle of c . Since $c' = \text{first}(\pi') \in Y^\sigma$, and Y^σ is closed under cyclic permutations (note that Y is closed under cyclic permutations if and only if $\neg Y$ is closed under cyclic permutations), we get that π is won by Player σ .

Details. It is enough to show that in the arena $A^{|S|}$, for every play π starting in w , there is a path π' starting in v , such that the first cycle of π' is a cyclic permutation of the first cycle $\pi_m \dots \pi_n$ of π . We construct π' as follows. Let ρ be a path

⁸ This result corrects a mistake in [4] that claims that this is a sufficient condition (note that they do not address the question of whether it is necessary).

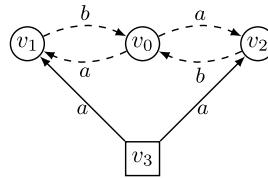


Fig. 3. Sample arena B . Circles are Player 0 nodes, squares are Player 1 nodes, solid lines are edges, dashed lines represent (sequences of edges that are) simple paths.

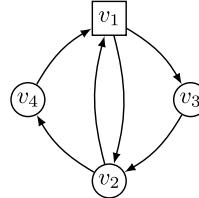


Fig. 4. Sample arena C . Circles are Player 0 nodes, squares are Player 1 nodes, solid lines are edges.

in $\mathcal{A}^{|S|}$ from v to w (such a path exists since $w \in \text{reach}(S, v)$). Let ℓ be the smallest index such that ρ_ℓ intersects $\pi[1, n]$, i.e., such that $\text{end}(\rho_\ell) = \text{start}(\pi_j)$ for some $j \leq n$. Note that ℓ is well defined since $\text{end}(\rho) = w = \text{start}(\pi_1)$. We consider two cases (depending on whether or not ρ first intersects π before π starts traversing $\text{first}(\pi)$).

Case $j \leq m$. Let $\pi' = \rho_1 \dots \rho_\ell \pi_j \dots \pi_n$, and note that $\text{first}(\pi') = \text{first}(\pi)$.

Case $m < j \leq n$. Let $\pi' = \rho_1 \dots \rho_\ell \pi_j \dots \pi_n \pi_m \dots \pi_{j-1}$, and note that $\text{first}(\pi') = \pi_j \dots \pi_n \pi_m \dots \pi_{j-1}$ which is a cyclic permutation of $\text{first}(\pi)$. \square

Theorem 3.

1. If Y is not closed under cyclic permutations then there is a game in $\text{FCG}(Y)$ that is not point-wise memoryless determined.
2. There exists a cycle property Y , closed under cyclic permutations, and a game in $\text{FCG}(Y)$ that is not point-wise memoryless determined.
3. If a cycle property Y is closed under cyclic permutations, then every game in $\text{FCG}(Y)$ that is point-wise memoryless for Player σ is also uniform memoryless for Player σ .

Proof. For the first item, assume that Y is not closed under cyclic permutations, and let $a, b \in \mathbb{U}^*$ be such that $ab \in Y$ but $ba \notin Y$. Consider the arena in Fig. 3. Observe that Player 0 wins from v_3 , but in order to do so he needs to remember if the play arrived to v_0 from v_1 or from v_2 .

For the second item, consider the arena in Fig. 4, and the cycle property $Y = \text{cyc-EvenLen}$. Obviously, Y is closed under cyclic permutations. However, starting at v_1 , Player 0 has a winning strategy, but no memoryless winning strategy – when choosing the outgoing edge from v_2 the player needs to remember if the previous node was v_1 (in which case he should return to v_1), or v_3 (in which case he should go to v_4).

For the third item, write $W_\sigma := \text{WR}^\sigma(\mathcal{A}, O)$, and assume that the game is point-wise memoryless for Player σ , and for every $v \in W_\sigma$ let S_v be a memoryless winning strategy for Player σ from v .

Sketch. The idea is to define a memoryless strategy S for Player σ that is winning from every node $v \in W_\sigma$ by considering the nodes of W_σ (in some arbitrary order), and if v is the next node in W_σ for which $S(v)$ is not yet defined, then define $S(w) := S_v(w)$ for all $w \in \text{Reach}(S_v, v)$ that have not yet been defined. Thus, S is memoryless by construction. The main work is to show that it is winning (this is where Lemma 4 is used).

Details. Fix some arbitrary linear ordering $v_1 \prec v_2 \prec \dots \prec v_n$ of the nodes in W_σ . We iteratively build a memoryless strategy S for Player σ that is winning from every node $v \in W_\sigma$. At each round $j \geq 0$ of this construction, we write $V^j \subseteq V$ for the set of nodes considered by the end of that round, and have that S is defined for every node in $V^j \cap W_\sigma$. At round 0, we begin with $V^0 = \emptyset$, and with $S(v)$ undefined for all $v \in V_\sigma$. At round $j > 0$, if $V^{j-1} = V$ then we are done, and otherwise we proceed as follows:

- (1) If $W_\sigma \not\subseteq V^{j-1}$, let v be the smallest vertex (by the ordering \prec) such that $v \in W_\sigma \setminus V^{j-1}$. Set $V^j := V^{j-1} \cup \text{reach}(S_v, v)$, and for every $w \in (V_\sigma \cap \text{reach}(S_v, v)) \setminus V^{j-1}$ define $S(w) := S_v(w)$.
- (2) If $W_\sigma \subseteq V^{j-1}$, then set $V^j = V$, and for every node $v \in V_\sigma \setminus V^{j-1}$ define $S(v)$ arbitrarily. Intuitively, moves from these nodes are unimportant since they are not reachable from any node in W_σ on any play consistent with S .

It is easy to see that S is well defined and memoryless.

It remains to show that S is winning from every $w \in W_\sigma$. The proof is by induction on the round number j . The induction hypothesis is that (i) if $w \in V^j$ then $\text{reach}(S, w) \subseteq V^j$ and; (ii) if $W_\sigma \not\subseteq V^{j-1}$ then S is winning from every $w \in V^j$.

Observe that, by taking the maximal j for which $W_\sigma \not\subseteq V^{j-1}$, item (ii) in the induction hypothesis implies that S is winning from every $w \in W_\sigma$.

For $j = 0$, the hypothesis is trivially true. Assume that the hypothesis holds for all $0 \leq l < j$, and consider round j . If $W_\sigma \subseteq V^{j-1}$, then (i) is true since the construction uses case (2) and sets $V^j = V$, and (ii) is trivially true. If, on the other hand, $W_\sigma \not\subseteq V^{j-1}$ (i.e., the construction uses cases (1)), then let $w \in \text{reach}(S_v, v)$ be some node added at round j . Note that to prove that (i) holds, it is enough to show that every node w' , that is reachable in $\mathcal{A}^{|S|}$ from w , was added before or at round j . In other words, we have to show that $\text{reach}(S, w) \subseteq (V^{j-1} \cup \text{reach}(S_v, v))$. This can be easily done by inducting on the length of the node-path $\rho = v_1 \dots v_k \in V^*$ from w to w' in $\mathcal{A}^{|S|}$. For $k = 0$ this is trivially true. Assume it is true for ρ of length k , and consider ρ of length $k + 1$. Now, if ρ_k was added at a previous round, then so did ρ_{k+1} by (i) in the induction hypothesis applied to ρ_k . Otherwise, ρ_k was added at this round (and thus $\rho_k \in \text{reach}(S_v, v)$), in which case if ρ_k belongs to the opponent (i.e., $\rho_k \in V_{1-\sigma}$) then all its successors, and in particular ρ_{k+1} , are in $\text{reach}(S_v, v)$; and if $\rho_k \in V_\sigma$ then $\rho_{k+1} = S(\rho_k) = S_v(\rho_k) \in \text{reach}(S_v, v)$. To complete the proof, we have to show that (ii) holds, i.e., that S is winning from w .

To see that S is winning from w , take any play $\pi \in \text{plays}(S, w)$, and let $\pi_m \dots \pi_n$ be the first cycle of π . There are two options: either the prefix $\pi_1 \dots \pi_n$ is consistent with S_v , or it isn't. If it is, since S_v is winning for Player σ from every node in $\text{reach}(S_v, v)$ (Lemma 4), and thus in particular from w , we have that π is won by Player σ . Assume then that $\pi_1 \dots \pi_n$ is not consistent with S_v . Note that by our choice of π it is consistent with S and thus, for some $1 \leq h < n$, we have that $S_v(\text{start}(\pi_h)) \neq \text{end}(\pi_h) = S(\text{start}(\pi_h))$. Let $k \leq n$ be the smallest index such that $\text{start}(\pi_k) \in V^{j-1}$. Such a k exists by the above inequality and the fact that (by construction) S agrees with S_v on all nodes added at round j . Observe that if $k > m$ then all the nodes appearing on $\pi_m \dots \pi_n$ are in $\text{reach}(S, \pi_k)$, simply by following the cycle $\pi_k \dots \pi_n \pi_m \dots \pi_{k-1}$. Hence, since by item (i) in the induction hypothesis $\text{reach}(S, \text{start}(\pi_k)) \subseteq V^{j-1}$, the minimality of k implies that $k \leq m$. We conclude that the suffix $\pi' = \pi_k \pi_{k+1} \dots$ of π , which is a play consistent with S starting from $\text{start}(\pi_k) \in V^{j-1}$, satisfies $\text{first}(\pi') = \text{first}(\pi)$. By item (ii) in the induction hypothesis, π' is won by Player σ , hence so is π , which completes the proof. \square

Theorems 2 and 3 give us the following corollary:

Corollary 1. *If a cycle property Y is closed under cyclic permutations, then every solitaire game in $\text{FCG}(Y)$ is uniform memoryless determined.*

5. Memoryless determinacy of first-cycle games

In this section we give a necessary and sufficient condition for a FCG to be memoryless determined. In Section 7 we give an easy-to check condition that is sufficient, but not necessary.

5.1. A full characterisation of memoryless determinacy of all games in $\text{FCG}(Y)$

We begin by introducing some useful shorthand notation. Given an arena $\mathcal{A} = (V_0, V_1, E, \mathbb{U}, \lambda)$, and a node $z \in V$, for a path $\pi \in E^* \cup E^\omega$, define $N_z(\pi) \in \mathbb{N} \cup \{\infty\}$ to be the index of the first edge that starts with z , if one exists. Formally, $N_z(\pi) := \infty$ if z does not occur on π , and otherwise $N_z(\pi) := \min\{j : \text{start}(\pi_j) = z\}$. Also, define $\text{head}_z(\pi) := \pi[1, N_z(\pi) - 1]$ to be the prefix of π before $N_z(\pi)$, and $\text{tail}_z(\pi) := \pi[N_z(\pi), |\pi|]$ to be the suffix of π starting at $N_z(\pi)$. By convention, if $N_z(\pi) = \infty$ then $\text{head}_z(\pi) = \pi$ and $\text{tail}_z(\pi) = \epsilon$.

We now define a game, that is very similar to the first-cycle game, except that one of the nodes of the arena is designated as a “reset” node:

Definition 1. Fix an arena \mathcal{A} , a vertex $z \in V$, and a cycle property Y . Define the objective $O_{z\text{-first}}^{\mathcal{A}}(Y) \subseteq \text{plays}(\mathcal{A})$ to consist of all plays π satisfying the following property: if $\text{head}_z(\pi)$ is not a simple path then $\text{first}(\pi) \in Y$, and otherwise $\text{first}(\text{tail}_z(\pi)) \in Y$.

Playing the game with objective $O_{z\text{-first}}(Y)$ is like playing the first-cycle game over Y , however, if no cycle is formed before reaching z for the first time, the prefix of the play up to that point is ignored. Thus, in a sense, the game is reset. Also note that if play starts from z , then the game reduces to a first-cycle game. It turns out that we may assume that a strategy of $(\mathcal{A}, O_{z\text{-first}}(Y))$ makes the same move every time it reaches z :

Definition 2 (*Forgetful at z from v*). For an arena \mathcal{A} , a vertex $v \in V$, a Player $\sigma \in \{0, 1\}$, and a vertex $z \in V_\sigma$ belonging to Player σ , we call a strategy T for Player σ *forgetful at z from v* if there exists $z' \in V$ such that $(z, z') \in E$ and for all $\pi \in \text{plays}(T, v)$, and all $n \in \mathbb{N}$, if $\text{start}(\pi_n) = z$ then $\text{end}(\pi_n) = z'$.

Lemma 5 (*Forgetful at z from v*). Fix an arena \mathcal{A} , a vertex $v \in V$, a Player $\sigma \in \{0, 1\}$, and a vertex $z \in V_\sigma$ belonging to Player σ . In the game $(\mathcal{A}, O_{z\text{-first}}(Y))$, if Player σ has a strategy S that is winning from v , then Player σ has a strategy T that is winning from v and that is forgetful at z from v .

Proof. We begin with the intuition.

Sketch. The second time z appears on a play, the winner is already determined, and so the strategy is free to repeat the first move it made at z . On the other hand, when a play visits z the first time, the strategy can make the same move regardless of the history of the play before z , because the winning condition ignores this prefix.

Details. We may suppose that z appears on some play of $\text{plays}(S, v)$, otherwise we can take T to be S . Let ρ be a simple path from v to z that is consistent with S . Let $z' := S(\rho)$. Define the strategy T as follows:

$$T(u) := \begin{cases} S(u) & \text{if } z \text{ does not appear on } u, \\ z' & \text{if } \text{end}(u) = z, \\ S(\rho \cdot \text{tail}_z(u)) & \text{otherwise.} \end{cases}$$

By definition, T is forgetful at z from v . It remains to show that every $\pi \in \text{plays}(T, v)$ is won by Player σ .

First consider the case that $\text{head}_z(\pi)$ is not a simple path. By definition, S and T agree on $\text{head}_z(\pi)$, and since S is winning, the first cycle on $\text{head}_z(\pi)$ (and thus also on π) satisfies Y^σ , and π is won by Player σ .

Now consider the case that $\text{head}_z(\pi)$ is a simple path. We need to show that $\text{first}(\text{tail}_z(\pi))$ is in Y^σ . Define $\pi' := \rho \cdot \text{tail}_z(\pi)$. It is easy to see that π' is consistent with T . We claim that $\pi' \in \text{plays}(S, v)$. Indeed, the prefix $\rho \cdot (z, z')$ is consistent with S ; and for every $j \geq |\rho| + 1$, such that $\text{end}(\pi'_j) \in V_\sigma$, we have, by the third case in the definition of T , that $T(\pi'[1, j]) = S(\rho \cdot \text{tail}_z(\pi'[1, j])) = S(\pi'[1, j])$ (the second equality holds since, by the definition of tail_z , $\rho \cdot \text{tail}_z(\pi'[1, j]) = \pi'[1, j]$). Now, since π' is consistent with T , we have that $T(\pi'[1, j]) = \text{end}(\pi'_{j+1})$, and thus $S(\pi'[1, j]) = \text{end}(\pi'_{j+1})$. This completes the proof of the claim.

To finish the proof, note that $\text{head}_z(\pi')$ is a simple path (it is ρ), and that $\text{tail}_z(\pi') = \text{tail}_z(\pi)$. Hence, since $\text{head}_z(\pi')$ is a simple path, and S is winning from v , we know that $\text{first}(\text{tail}_z(\pi'))$ satisfies Y^σ . Thus, since $\text{head}_z(\pi)$ is a simple path and $\text{first}(\text{tail}_z(\pi))$ satisfies Y^σ , we can conclude that π is won by Player σ . \square

We now define the basic notion behind our necessary and sufficient condition for games in $\text{Fcg}(Y)$ to be memoryless determined.

Definition 3. For an arena \mathcal{A} , and a node v in \mathcal{A} , say that \mathcal{A} is Y -resettable from v if for every node z , the same player wins from v in both $(\mathcal{A}, O_{\text{first}}(Y))$ and $(\mathcal{A}, O_{z\text{-first}}(Y))$.

First, we show that Y -resetability is a sufficient condition for having memoryless strategies.

Theorem 4. Given an arena \mathcal{A} , if a node v is such that every sub-arena of \mathcal{A} is Y -resettable from v , then the game $(\mathcal{A}', O_{\text{first}}(Y))$ is memoryless from v for every sub-arena \mathcal{A}' of \mathcal{A} .

Proof. A node $z \in V$ is a *choice node* of an arena $\mathcal{B} = (V_0, V_1, E^{\mathcal{B}}, \mathbb{U}, \lambda)$, if there are at least two distinct vertices $v', v'' \in V$ such that $(z, v') \in E^{\mathcal{B}}$ and $(z, v'') \in E^{\mathcal{B}}$.

Sketch. Fix a sub-arena \mathcal{A}' of \mathcal{A} . Suppose Player σ has a winning strategy from v in \mathcal{A}' . We induct on the number of choice nodes of Player σ . Let z be a choice node for Player σ (if there are none, the result is immediate). Since \mathcal{A}' is Y -resettable from v , Player σ also wins the game with objective $O_{z\text{-first}}(Y)$ from v . By Lemma 5, Player σ has a strategy S that is winning from v and that is also forgetful at z from v . Thus we may form a sub-arena \mathcal{B} of \mathcal{A}' (and hence of \mathcal{A}) by removing all edges from z that are not taken by S . Observe that S is winning from v in $(\mathcal{B}, O_{z\text{-first}}(Y))$. Since the sub-arena \mathcal{B} is Y -resettable from v , Player σ also wins $(\mathcal{B}, O_{\text{first}}(Y))$ from v . But \mathcal{B} has less choice nodes for Player σ , and thus, by induction, Player σ has a memoryless winning strategy from v in $(\mathcal{B}, O_{\text{first}}(Y))$. This memoryless strategy is also winning from v in \mathcal{A}' .

Details. Fix a sub-arena \mathcal{A}' of \mathcal{A} , and let $\sigma \in \{0, 1\}$ be such that $v \in \text{WR}^\sigma(\mathcal{A}', O_{\text{first}}(Y))$. The n th inductive hypothesis states: for every sub-arena \mathcal{B} (of \mathcal{A}'), in which Player σ has exactly n choice nodes, if Player σ has a winning strategy from v in $(\mathcal{B}, O_{\text{first}}(Y))$, then Player σ has a *memoryless* winning strategy from v in $(\mathcal{B}, O_{\text{first}}(Y))$.

The base case is when $n = 0$, i.e., Player σ has no choice nodes in \mathcal{B} . In this case, Player σ has a single strategy: given a history $u \in H_\sigma(\mathcal{B})$ with $\text{end}(u) = w$, then take the unique edge $(w, w') \in E^{\mathcal{B}}$. Note that this strategy is memoryless.

Let $n > 0$, and suppose the inductive hypothesis holds for $n - 1$. We will prove it holds for n . To this end, let $\mathcal{B} = (V_0, V_1, E^{\mathcal{B}}, \mathbb{U}, \lambda)$ be a sub-arena (of \mathcal{A}') with n choice nodes for Player σ . Fix one such choice node, and call it z . Suppose Player σ has a winning strategy (not necessarily memoryless) from v in $(\mathcal{B}, O_{\text{first}}(Y))$. Since, by assumption, \mathcal{B} is Y -resettable from v , there is also a winning strategy S for Player σ from v in $(\mathcal{B}, O_{z\text{-first}}(Y))$. We will use S to prove Player σ has a memoryless winning strategy from v in $(\mathcal{B}, O_{\text{first}}(Y))$. By Lemma 5, we may assume that S is forgetful at

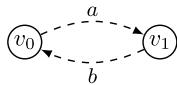


Fig. 5. The dashed lines represent simple paths.

z from v , i.e., there exists $z' \in V$ such that $(z, z') \in E^{\mathcal{B}}$ and for all $\pi \in \text{plays}_{\mathcal{B}}(S, v)$, and all $i \in \mathbb{N}$, if $\text{start}(\pi_i) = z$ then $\text{end}(\pi_i) = z'$.

Define the sub-arena \mathcal{B}_z to be the same as \mathcal{B} but with all edges out of z removed except for (z, z') . That is, $\mathcal{B}_z := (V_0, V_1, E_z, \mathbb{U}, \lambda')$ where $E_z := E^{\mathcal{B}} \setminus \{(z, x) : x \neq z'\}$ and λ' is λ restricted to E_z . Then S is well-defined on \mathcal{B}_z (i.e., by our assumption it never says to move from z to a node other than z'). Since S is winning for Player σ from v in the game $(\mathcal{B}, O_{z\text{-first}}(Y))$ and node z belongs to Player σ , conclude that S is winning for Player σ from v in $(\mathcal{B}_z, O_{z\text{-first}}(Y))$. Being a sub-arena of \mathcal{A} , by assumption, the arena \mathcal{B}_z is Y -resettable from v . Hence, Player σ wins from v in $(\mathcal{B}_z, O_{\text{first}}(Y))$. Since \mathcal{B}_z is a sub-arena of \mathcal{A}' and has $n - 1$ choice nodes for Player σ , we can apply the induction hypothesis to \mathcal{B}_z and obtain that Player σ has a memoryless strategy S_{mem} winning from v in $(\mathcal{B}_z, O_{\text{first}}(Y))$. Recall that \mathcal{B}_z was obtained from \mathcal{B} by removing outgoing edges from $z \in V_{\sigma}$ (i.e., by providing Player σ with less freedom of movement), and conclude that S_{mem} must be winning for Player σ from v in the game $(\mathcal{B}, O_{\text{first}}(Y))$. This completes the inductive step. \square

It is worth noting that the assumption in [Theorem 4](#), that every sub-arena of \mathcal{A} is Y -resettable from v , cannot be replaced by the weaker requirement that only \mathcal{A} is Y -resettable from v . Consider for example the arena \mathcal{A} in [Fig. 3](#) (page 10), and a cycle property $Y \subseteq \mathbb{U}^*$, such that $ab \in Y$ but $ba \notin Y$, for some $a, b \in \mathbb{U}^*$. As argued in [Theorem 3](#), the game $(\mathcal{A}, O_{\text{first}}(Y))$ is not memoryless starting at v_3 . While the sub-arena \mathcal{A}' , obtained by dropping the edge (v_0, v_1) , is not Y -resettable from v_3 (since for $z = v_0$ Player 0 wins $(\mathcal{A}', O_{z\text{-first}}(Y))$ but $(\mathcal{A}', O_{\text{first}}(Y))$ is won by Player 1) the arena \mathcal{A} is Y -resettable from v_3 .⁹

Before we provide a converse for [Theorem 4](#), we show that an additional assumption, namely that Y is closed under cyclic permutations, is needed:

Lemma 6. *If Y is not closed under cyclic permutations then there is an arena \mathcal{A} , and a node v , for which the game $(\mathcal{A}', O_{\text{first}}(Y))$ is memoryless from v for every sub-arena \mathcal{A}' of \mathcal{A} , but \mathcal{A} is not Y -resettable from v .*

Proof. Assume that Y is not closed under cyclic permutations, and let $a, b \in \mathbb{U}^*$ be such that $ab \in Y$ but $ba \notin Y$. Consider the arena in [Fig. 5](#), containing only Player 0 nodes, where the path from v_0 to v_1 is labelled by a , and the path from v_1 to v_0 is labelled by b . Observe that \mathcal{A} has only one sub-arena (itself), and there is a single strategy possible in the game $(\mathcal{A}, O_{\text{first}}(Y))$, and it is memoryless. However, \mathcal{A} is not Y -resettable from v_0 since, starting at v_0 , Player 0 wins the game $(\mathcal{A}, O_{\text{first}}(Y))$ but not the game $(\mathcal{A}, O_{z\text{-first}}(Y))$ for $z = v_1$. \square

We now prove that if Y is closed under cyclic permutations then the converse of [Theorem 4](#) is also true.

Theorem 5. *Let Y be closed under cyclic permutations, let \mathcal{A} be an arena, and let v be a node such that the game $(\mathcal{B}, O_{\text{first}}(Y))$ is memoryless from v for every sub-arena \mathcal{B} of \mathcal{A} . Then every sub-arena \mathcal{B} of \mathcal{A} is Y -resettable from v .*

Proof. We prove that, for every arena \mathcal{B} and a node v in it, if $(\mathcal{B}, O_{\text{first}}(Y))$ is memoryless from v then \mathcal{B} is Y -resettable from v . The theorem follows by taking \mathcal{B} to be any sub-arena of \mathcal{A} .

Let z be a node in \mathcal{B} . [Lemma 3](#) (Page 6) implies that FCGs are determined. Thus, it is enough to show that if Player σ wins from v in the game $(\mathcal{B}, O_{\text{first}}(Y))$, then he wins from v in the game $(\mathcal{B}, O_{z\text{-first}}(Y))$. So, fix a Player $\sigma \in \{0, 1\}$ and assume that Player σ wins from v in $(\mathcal{B}, O_{\text{first}}(Y))$. Since by our assumption this game is memoryless determined, there is a memoryless strategy S for Player σ winning from v in $(\mathcal{B}, O_{\text{first}}(Y))$. Consider the sub-arena $\mathcal{B}^{\parallel S}$ induced by S , and recall that every path in $\mathcal{B}^{\parallel S}$ is consistent with S . We claim that every simple cycle $\pi = \pi_1 \dots \pi_k$ (of some length k) in $\mathcal{B}^{\parallel S}$, that is reachable from v , satisfies Y^{σ} . Let ρ be a path in $\mathcal{B}^{\parallel S}$ of minimal length that starts in v and ends in $\text{start}(\pi_i)$ for some $1 \leq i \leq k$. Consider the path $\pi' = \rho \pi_i \dots \pi_k \pi_1 \dots \pi_{i-1}$. Since $\pi' \in \text{plays}_{\mathcal{B}}(S, v)$, and S is winning from v , we have that the first cycle c on π' satisfies Y^{σ} . Observe that (by our choice of ρ) $c = \pi_i \dots \pi_k \pi_1 \dots \pi_{i-1}$, and is thus a cyclic permutation of π . Since Y^{σ} is closed under cyclic permutations (recall that if Y is closed under cyclic permutations then so is $\neg Y$) then π satisfies Y^{σ} , and the claim is true. In other words, for every play $\rho' \in \text{plays}_{\mathcal{B}}(S, v)$, and every simple cycle $c' = \rho'_i \dots \rho'_j$ (for some $i, j \in \mathbb{N}$) on ρ' , we have that c' satisfies Y^{σ} . Hence, S is also winning from v in the game $(\mathcal{B}, O_{z\text{-first}}(Y))$. \square

If an arena \mathcal{A} is Y -resettable from v , for every node v , then we simply say that it is Y -resettable. In other words:

⁹ One can also come up with such an example (albeit a more complicated one) for $Y = \text{cyc-EvenLen}$ which is closed under cyclic permutations.

Definition 4. An arena \mathcal{A} is Y -resettable if for every $\sigma \in \{0, 1\}$, and every node z , we have that $\text{WR}^\sigma(\mathcal{A}, O_{z\text{-first}}(Y)) = \text{WR}^\sigma(\mathcal{A}, O_{\text{first}}(Y))$.

We conclude with this full characterisation:

Theorem 6 (Memoryless determinacy characterisation of FCGs). The following are equivalent for every cycle property Y :

1. Y is closed under cyclic permutations, and every arena \mathcal{A} is Y -resettable.
2. Every game in $\text{FCG}(Y)$ is uniform memoryless determined.

Proof. Suppose Y is closed under cyclic permutations. [Theorem 3](#) (part 3) says that every game in $\text{FCG}(Y)$ that is point-wise memoryless determined is uniform memoryless determined. But since every arena is assumed Y -resettable from every v , [Theorem 4](#) implies that every game in $\text{FCG}(Y)$ is point-wise memoryless determined.

Conversely, suppose every game in $\text{FCG}(Y)$ is uniform memoryless determined. By [Theorem 3](#) (part 1), Y must be closed under cyclic permutations. This also means we can apply [Theorem 5](#), and conclude that every arena is Y -resettable. \square

6. Strategy Transfer Theorem: infinite-duration cycle games and first-cycle games

In this section we define the connection between first-cycle games and games of infinite duration (such as parity games, etc.), namely the concept of Y -greedy games. We then prove the Strategy Transfer Theorem, which says, roughly, that for every arena, the winning regions of the First-Cycle Game over Y and a Y -greedy game coincide, and that memoryless winning strategies transfer between these two games.

Definition 5 (Greedy). Say that a game (\mathcal{A}, O) is Y -greedy if

$$O_{\text{all}}^{\mathcal{A}}(Y) \subseteq O \text{ and } O_{\text{all}}^{\mathcal{A}}(\neg Y) \subseteq E^\omega \setminus O.$$

Intuitively, a game (\mathcal{A}, O) is Y -greedy means that Player 0 can win the game (\mathcal{A}, O) if he ensures that every cycle in the cycles-decomposition of the play is in Y , and Player 1 can win if she ensures that every cycle in the cycles-decomposition of the play is not in Y .

An equivalent formulation is that (\mathcal{A}, O) is Y -greedy if

$$O_{\text{all}}^{\mathcal{A}}(Y) \subseteq O \subseteq O_{\text{exist}}^{\mathcal{A}}(Y),$$

where $O_{\text{exist}}^{\mathcal{A}}(Y)$ consists of all plays π such that some cycle in $\text{cycles}(\pi)$ satisfies Y .

Here are some examples.

1. Every all-cycles game $(\mathcal{A}, O_{\text{all}}(Y))$ is Y -greedy.
2. Every parity game is cyc-Parity-greedy.
3. Every game with ν -mean-payoff winning condition is cyc-MeanPayoff $_\nu$ -greedy.

Before proving the Strategy Transfer Theorem we need a lemma that states that one can pump a strategy S that is winning for the first-cycle game to get a strategy S^\circlearrowright that is winning for the all-cycles game by following S until a cycle is formed, removing that cycle from the history, and continuing. The fact that every winning strategy in the first-cycle game of Y can be pumped to obtain a winning strategy in a Y -greedy game, is why we call such games “greedy”.

Recall from the Definitions that for a finite path $\pi \in E^*$, the stack content at the end of $\text{CycDEC}(\epsilon, \pi)$ ([Algorithm 1](#)) is denoted $\text{stack}(\pi)$. Recall our notational abuse (Page 4) that for a finite path ρ , we may write $S(\rho)$ instead of $S(\text{nodes}(\rho))$.

Definition 6 (Pumping strategy). Fix an arena \mathcal{A} , a Player $\sigma \in \{0, 1\}$, and a strategy S for Player σ . Let the pumping strategy of S be the strategy S^\circlearrowright for Player σ , defined, on any input $u = v_1 \dots v_k \in H_\sigma(\mathcal{A})$, as follows:

- a. $S^\circlearrowright(u) := S(v_k)$ if $k = 1$ or $\text{stack}(\pi) = \epsilon$, and otherwise
- b. $S^\circlearrowright(u) = S(\text{stack}(\pi))$,

where $\pi \in E^*$ is the path corresponding to u , i.e., $\text{nodes}(\pi) = u$.

Note that S^\circlearrowright is well-defined since if $\text{stack}(\pi) \neq \epsilon$ then $\text{stack}(\pi)$ ends with $v_k \in V_\sigma$ and so $\text{stack}(\pi)$ is in the domain of S . Also note that if S is memoryless then the pumping strategy $S^\circlearrowright = S$.

Lemma 7. Let \mathcal{A}, σ, S and S^\circlearrowright be as in [Definition 6](#), and let $v \in V$. If $\pi \in \text{plays}(S^\circlearrowright, v)$ then for every cycle C in $\text{cycles}(\pi)$ there exists a finite path ρ consistent with S and starting with v such that:

- The first cycle on ρ is C (thus, in particular, if S is winning from v in the game $(\mathcal{A}, O_{\text{first}}(Y))$ then C satisfies Y^σ);
- ρ only contains edges from π .

Proof. Sketch. The strategy S^\circlearrowleft says to follow S , and when a cycle is popped by CycDEC, remove that cycle from the history and continue. Thus, for every cycle C that is popped, let l be the time at which the first edge of C is being pushed, and note that the stack up to time l followed by C is a path consistent with S whose first cycle is C .

Details. Fix $\pi \in \text{plays}(S^\circlearrowleft, v)$. Every cycle $C \in \text{cycles}(\pi)$ was output by the run of CycDEC(ϵ, π) at some time $j \in \mathbb{N}$, and is thus of the form $C = \text{cycle}^j(\pi)$. Let $\rho := \text{stack}_{j-1}(\pi) \cdot \pi_j$, and observe that C is the first cycle on ρ , and the second item in the statement of the lemma is true.

For the first item, it is sufficient to prove, for all $k \in \mathbb{N}$, that $\text{stack}_{k-1}(\pi) \cdot \pi_k$ is consistent with S and starts with v . We remind the reader that, by definition, ρ is consistent with S iff:

1. $S(\text{start}(\rho_1)) = \text{end}(\rho_1)$ if $\text{start}(\rho_1) \in V_\sigma$, and
2. $S(\rho[1, n]) = \text{end}(\rho_{n+1})$ for $1 \leq n < |\rho|$ with $\text{end}(\rho[1, n]) \in V_\sigma$.

We prove that $\text{stack}_{k-1}(\pi) \cdot \pi_k$ is consistent with S and starts with v , by induction on $k \in \mathbb{N}$.

The base is when $k = 1$. Since $\text{stack}_0(\pi) = \epsilon$, we show that the path consisting of the single edge π_1 , which starts with v by definition, is consistent with S . Note that we are in item 1. of the consistency condition with $\rho = \pi_1$. So suppose $\text{start}(\pi_1) \in V_\sigma$. Then:

$$\begin{aligned} \text{end}(\pi_1) &= S^\circlearrowleft(\text{start}(\pi_1)) && \text{since } \pi \text{ is consistent with } S^\circlearrowleft, \\ &= S(\text{start}(\pi_1)) && \text{by definition of } S^\circlearrowleft, \text{ part a).} \end{aligned}$$

For the inductive step, let $k > 1$ and suppose the inductive hypothesis holds for $k - 1$. We prove it holds for k . There are two cases.

i) Suppose $\text{stack}_{k-1}(\pi) = \epsilon$. To show π_k is consistent with S , note that we are again in item 1. of the consistency condition, but this time with $\rho = \pi_k$. Repeat the argument in the base case with π_k replacing π_1 . To show that π_k starts with v argue as follows. The fact that $\text{stack}_{k-1}(\pi) = \epsilon$ means that after pushing π_{k-1} onto the stack during step $k - 1$ of the algorithm CycDEC(ϵ, π), the resulting stack forms a cycle. In other words, $\text{end}(\pi_{k-1}) = \text{start}(\text{stack}_{k-2}(\pi) \cdot \pi_{k-1})$. But $\text{start}(\pi_k) = \text{end}(\pi_{k-1})$ since π is a path, and $\text{start}(\text{stack}_{k-2}(\pi) \cdot \pi_{k-1}) = v$ by the induction hypothesis. Thus $\text{start}(\pi_k) = v$.

ii) Suppose $\text{stack}_{k-1}(\pi) \neq \epsilon$. The induction hypothesis states that the path $\text{stack}_{k-2}(\pi) \cdot \pi_{k-1}$ is consistent with S and starts with v . Observe that $\text{stack}_{k-1}(\pi)$ is a (non-empty) prefix of $\text{stack}_{k-2}(\pi) \cdot \pi_{k-1}$. So, $\text{stack}_{k-1}(\pi)$ starts with v , and, being a prefix of a path consistent with S , is consistent with S . Thus we only need to consider π_k . That is, we are in item 2. of the consistency condition, with $\rho = \text{stack}_{k-1}(\pi) \cdot \pi_k$ and $n = |\text{stack}_{k-1}(\pi)|$. If $\text{end}(\rho[1, n]) \in V_\sigma$ then

$$\begin{aligned} \text{end}(\rho_{n+1}) &= \text{end}(\pi_k) && \text{since } \rho_{n+1} = \pi_k, \\ &= S^\circlearrowleft(\pi[1, k-1]) && \text{since } \pi \text{ is consistent with } S^\circlearrowleft, \\ &= S(\text{stack}_{k-1}(\pi)) && \text{by definition of } S^\circlearrowleft, \text{ part b),} \\ &= S(\rho[1, n]) && \text{since } \rho[1, n] = \text{stack}_{k-1}(\pi). \end{aligned}$$

This completes the inductive step and the proof. \square

Corollary 2. Fix Player $\sigma \in \{0, 1\}$ and let (\mathcal{A}, O) be a Y -greedy game. If S is a strategy for Player σ that is winning from v in $(\mathcal{A}, O_{\text{first}}(Y))$ then S^\circlearrowleft is winning from v in (\mathcal{A}, O) .

Proof. Suppose S is winning from v in the game $(\mathcal{A}, O_{\text{first}}(Y))$. Then, by Lemma 7, for every play $\pi \in \text{plays}(S^\circlearrowleft, v)$, every cycle in $\text{cycles}(\pi)$ satisfies Y^σ , i.e., $\pi \in O_{\text{all}}^{\mathcal{A}}(Y^\sigma)$. By definition of Y -greedy, this means that S^\circlearrowleft is winning from v in the game (\mathcal{A}, O) . \square

We now have the ingredients for the proof of the Strategy Transfer Theorem:

Theorem 7 (Strategy transfer). Let (\mathcal{A}, O) be a Y -greedy game, and let $\sigma \in \{0, 1\}$.

1. The winning regions for Player σ in the games (\mathcal{A}, O) and $(\mathcal{A}, O_{\text{first}}(Y))$ coincide.
2. For every memoryless strategy S for Player σ , and vertex $v \in V$ in arena \mathcal{A} : S is winning from v in the game (\mathcal{A}, O) if and only if S is winning from v in the game $(\mathcal{A}, O_{\text{first}}(Y))$.

Proof. Let $Y \subseteq \mathbb{U}^*$ be a cycle property and \mathcal{A} an arena. Suppose that (\mathcal{A}, O) is Y -greedy. We begin by proving the first item. Use [Corollary 2](#) to get that for $\sigma \in \{0, 1\}$,

$$\text{WR}^\sigma(\mathcal{A}, O_{\text{first}}(Y)) \subseteq \text{WR}^\sigma(\mathcal{A}, O).$$

Since first-cycle games are determined ([Lemma 3](#)), the winning regions $\text{WR}^0(\mathcal{A}, O_{\text{first}}(Y))$ and $\text{WR}^1(\mathcal{A}, O_{\text{first}}(Y))$ partition V . Thus, since $\text{WR}^0(\mathcal{A}, O)$ and $\text{WR}^1(\mathcal{A}, O)$ are disjoint, the containments above are equalities, as required for item 1.

We prove the second item. Since $S = S^\circlearrowleft$ if S is memoryless, conclude by [Corollary 2](#): if S is winning from v in the game $(\mathcal{A}, O_{\text{first}}(Y))$ then it is winning from v in the game (\mathcal{A}, O) . For the other direction, assume by contraposition that S is not winning from v in the game $(\mathcal{A}, O_{\text{first}}(Y))$. Since S is memoryless, plays of \mathcal{A} consistent with S are exactly infinite paths in the induced sub-arena $\mathcal{A}^{\parallel S}$. Hence, there is a path π in the induced solitaire arena $\mathcal{A}^{\parallel S}$ for which the first cycle, say $\pi[i, j]$, satisfies $Y^{1-\sigma}$. Define the infinite path $\pi' := \pi[1, i-1] \cdot (\pi[i, j])^\omega$ and note that, being a path in $\mathcal{A}^{\parallel S}$, it is a play of \mathcal{A} consistent with S . Moreover, π' has the property that every cycle in its cycles-decomposition (i.e., $\pi[i, j]$) satisfies $Y^{1-\sigma}$. Since (\mathcal{A}, O) is Y -greedy, S is not winning from v in the game (\mathcal{A}, O) . \square

Since FCGs are determined ([Lemma 3](#), Page 6) use [Theorem 7](#) to get:

Corollary 3. Every Y -greedy game (\mathcal{A}, O) is determined, has the same winning regions as $(\mathcal{A}, O_{\text{first}}(Y))$, and is point-wise (uniform) memoryless determined if and only if the game $(\mathcal{A}, O_{\text{first}}(Y))$ is point-wise (uniform) memoryless determined.

We now state our main result concerning Y -greedy games.

Theorem 8 (Memoryless determinacy characterisation of greedy games). For every cycle property Y , the following are equivalent:

1. Y is closed under cyclic permutations, and every arena is Y -resettable.
2. Every Y -greedy game is uniform memoryless determined.

Proof. This follows from [Corollary 3](#), [Theorem 6](#), and the fact that for every arena \mathcal{A} there is a Y -greedy game with arena \mathcal{A} , for example, the game $(\mathcal{A}, O_{\text{all}}(Y))$. \square

7. An easy to check sufficient condition on Y ensuring memoryless determinacy of FCGs

As we have seen Y -resetability together with closure under cyclic permutations provides a full characterization of those cycle properties Y for which the games in $\text{FCG}(Y)$, as well as any Y -greedy games, are memoryless determined. Even though, in many cases, checking whether Y is such that every arena \mathcal{A} (or just the arena(s) of interest) is Y -resettable is not too difficult, we believe that in practice it is much easier to use the following condition on Y which avoids reasoning about arenas altogether. The condition we suggest is the following:

$$Y \text{ and } \neg Y \text{ are closed under concatenation.}$$

As we later show, this condition (together with closure under cyclic permutations) ensures that every arena \mathcal{A} is Y -resettable, and thus by [Theorem 6](#), that all games in $\text{FCG}(Y)$ are memoryless determined. On the other hand, as we discuss in [Section 8](#), there is a cycle property Y which is closed under cyclic permutations and for which every arena \mathcal{A} is Y -resettable, even though Y does not satisfy the condition. Thus, this condition cannot replace Y -resetability as a necessary condition for memoryless determinacy of all games in $\text{FCG}(Y)$. However, it is applicable in a wide variety of cases, and is usually very easy to check. Consider for example the cycle properties given in [Section 2](#). For each of these properties Y , while proving that every arena is Y -resettable may not be hard, checking whether Y satisfies the condition above is almost completely trivial. Note that cyc-EvenLen, which is the only property among these which is closed under cyclic permutations but fails to satisfy this condition, would also fail to satisfy any other condition that guarantees memoryless determinacy since it actually admits FCGs that are not memoryless determined (cf. [Theorem 3](#), Page 10).

Our goal in the rest of this section is to prove the following theorem:

Theorem 9 (Easy to check). Let $Y \subseteq \mathbb{U}^*$ be a cycle property. If Y is closed under cyclic permutations, and both Y and $\neg Y$ are closed under concatenation, then every arena \mathcal{A} is Y -resettable.

By [Theorem 6](#) we get:

Corollary 4. Let $Y \subseteq \mathbb{U}^*$ be a cycle property. If Y is closed under cyclic permutations, and both Y and $\neg Y$ are closed under concatenation, then every game in $\text{FCG}(Y)$ is uniform memoryless determined.

Remark 2. Consider the cycle property $Y = \text{cyc-GoodForEnergy}$ (recall from the definitions this means that either the energy level is positive, or it is zero and the largest priority occurring is even). Note that Y is closed under cyclic permutations, and both Y and $\neg Y$ are closed under concatenation. Conclude that every game in $\text{FCG}(Y)$ is pointwise-memoryless determined. This fact allows one to obtain a proof of Lemma 4 in [5] that no longer relies on the incorrect result from [4].

We begin by introducing a new kind of objective $O_{\text{tail}}^{\mathcal{A}}(Y)$:

Definition 7. For an arena \mathcal{A} and cycle property Y , define the objective $O_{\text{tail}}^{\mathcal{A}}(Y)$ to consist of all plays π such that there is a suffix π' of π with the property that every cycle in $\text{cycles}(\pi')$ satisfies Y .

Note that this is not the same as saying that $\lambda(C) \in Y$ for all but finitely many cycles C in $\text{cycles}(\pi)$.¹⁰

Definition 8. An arena \mathcal{A} is Y -unambiguous if $O_{\text{tail}}^{\mathcal{A}}(Y) \cap O_{\text{tail}}^{\mathcal{A}}(\neg Y) = \emptyset$.

In other words, \mathcal{A} is Y -unambiguous if no play in \mathcal{A} has two suffixes, π, π' such that every cycle in the cyclic decomposition of π is in Y , and every cycle in the cyclic decomposition of π' is not in Y .

In order to prove [Theorem 9](#) we have to show that, for Y satisfying the assumptions of the theorem, for every arena \mathcal{A} , every Player $\sigma \in \{0, 1\}$, and every node z in \mathcal{A} , we have that $\text{WR}^{\sigma}(\mathcal{A}, O_{\text{first}}(Y)) = \text{WR}^{\sigma}(\mathcal{A}, O_{z\text{-first}}(Y))$. The proof is in three parts:

- Part 1. If Y is closed under cyclic permutations, and both Y and $\neg Y$ are closed under concatenation, then every arena \mathcal{A} is Y -unambiguous.
- Part 2. If \mathcal{A} is Y -unambiguous then $\text{WR}^{\sigma}(\mathcal{A}, O_{\text{first}}(Y)) = \text{WR}^{\sigma}(\mathcal{A}, O_{\text{tail}}(Y))$.
- Part 3. If \mathcal{A} is Y -unambiguous then $\text{WR}^{\sigma}(\mathcal{A}, O_{z\text{-first}}(Y)) = \text{WR}^{\sigma}(\mathcal{A}, O_{\text{tail}}(Y))$.

Observe that parts 2 and 3 imply that if an arena \mathcal{A} is Y -unambiguous then it is Y -resettable.

We first need a couple of definitions and a few easy lemmas. Say that Y is *closed under insertions* if it is closed under concatenation and: $ac \in Y$ and $b \in Y$ imply that $abc \in Y$, for all $a, b, c \in \mathbb{U}^*$. The *closure of Y under insertions*, is defined to be the smallest subset of \mathbb{U}^* (with respect to set containment) that contains Y and is closed under insertions.

Lemma 8. If Y is closed under cyclic permutations and under concatenation then Y is closed under insertions.

Proof. Assume that $ac, b \in Y$. We have that $ac \in Y \implies ca \in Y \implies cab \in Y \implies abc \in Y$. The middle implication is since Y is closed under concatenation, and the other two implications are since Y is closed under cyclic permutations. \square

Fix an arena $\mathcal{A} = (V_0, V_1, E, \mathbb{U}, \lambda)$. Given a simple path $s \in E^*$, and a path $u \in E^*$ such that $\text{end}(s) = \text{start}(u)$, write $\text{labels}(s, u) = \{\lambda(C) \mid C \in \text{cycles}(s, u)\}$ for the set of the labels of the cycles output by $\text{CycDEC}(s, u)$.

Lemma 9. Given an arena $\mathcal{A} = (V_0, V_1, E, \mathbb{U}, \lambda)$, let s, u be paths in \mathcal{A} such that u is a cycle and $\text{end}(s) = \text{start}(u) = v$. We have that:

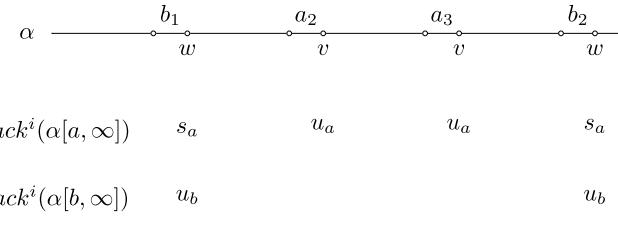
1. if $\text{stack}(s, u) = s$ then the insertion closure of $\text{labels}(s, u)$ contains $\lambda(u)$;
2. if v is the only node that appears on both s and u then $\text{stack}(s, u) = s$.

Proof. Sketch. For the first item, note that the assumption $\text{stack}(s, u) = s$ implies that the cycles in $\text{cycles}(s, u)$ contain exactly the edges of u . Hence, it is not hard to see that $\lambda(u)$ is in the insertion closure of $\text{labels}(s, u)$. Intuitively, the algorithm output the cycles in $\text{cycles}(s, u)$ by “popping them out” of u . Thus, we can reverse this process and reconstruct u using insertion operations on the elements of $\text{cycles}(s, u)$.

Details. More formally, let C^1, \dots, C^m be the elements of $\text{cycles}(s, u)$ in the order they were output. We iteratively construct a string $w \in E^*$ until we get $w = u$. We start with w being the empty string, and count down from m to 1. At step i in the count, we insert the cycle C^i into the correct position in w , as follows: let j be such that $u_j = C_0^i$ (i.e., the first edge of C^i), and set $w := w_1 \dots w_h \cdot C^i \cdot w_{h+1} \dots w_{|w|}$, where h is the maximal index such that all the edges w_1, \dots, w_h are in $\{u_1, \dots, u_{j-1}\}$.

It is easy to see that when the construction ends w contains exactly the edges appearing in C^1, \dots, C^m and thus, as noted before, exactly the edges of u . We claim that the edges are also ordered correctly (i.e., if $a < b$ then w_a appears in w before w_b) and thus, $w = u$ as required. Assume by way of contradiction that the correct ordering in w was violated for the first time when C^i was inserted. Let u_j be the first edge of C^i and h be the position where C^i was inserted into w , as defined before. Recall that after the insertion the constructed string is $w_1 \dots w_h \cdot C^i \cdot w_{h+1} \dots w_{|w|}$. Observe that since all edges of C_i are internally ordered correctly, and all of them correctly appear (by our choice of h) after $w_1 \dots w_h$, the only possible violation is that some edge u_t in C^i incorrectly appears before the edge $w_{h+1} = u_r$, i.e., that $j < r < t$. Also note that at the step where C^i was output, all edges above u_j that were on the stack were popped, and all edges up to u_t were already processed. Hence, it can not be that the edge u_r was output by the algorithm after C^i , which is a contradiction to the fact that u_r is already in w before the insertion of C^i . This completes the proof of the claim.

¹⁰ For instance, consider the arena in [Fig. 4](#), take Y to be cyc-EvenLen, and let $\pi := [(v_1, v_2)(v_2, v_1)(v_1, v_3)(v_3, v_2)(v_2, v_4)(v_4, v_1)]^\omega$. Note that i) decomposing the suffix π' starting with the second edge results in all cycles having odd length, and ii) it is not the case that almost all cycles in the cycles-decomposition of π (i.e., starting with the first edge) have odd length – in fact, they all have even length.

**Fig. 6.** Visualisation of the processing of the play α .

For the second item, observe that the assumption made there implies that, while processing u , the algorithm $\text{CycDEC}(s, u)$ cannot pop any edge in s . Hence, $\text{stack}(s, u) = s$ does not hold only if there exists j such that the edge $u_j = (v, v')$ is pushed on top of s but never popped. Observe that $v' \neq v = \text{src}(u) = \text{trg}(u)$ (the first inequality is since a self-loop is always popped, the rest by our assumption that u is a cycle that starts in v), and thus u_j is not the last edge of u . But this is a contradiction since the (non-empty) suffix $u_{j+1} \dots u_{|u|}$ of u contains at least one edge e' with $\text{end}(e') = v$ (namely $u_{|u|}$), and the algorithm would have popped the edge u_j when it encountered the first edge that ends in v . \square

We are now ready to show part 1 in the proof of [Theorem 9](#).

Proposition 2. *Let $Y \subseteq \mathbb{U}^*$ be a cycle property. If Y is closed under cyclic permutations, and both Y and $\neg Y$ are closed under concatenation, then every arena \mathcal{A} is Y -unambiguous.*

Proof. First, note that since Y is closed under cyclic permutations then so is $\neg Y$. By [Lemma 8](#), we have that Y , as well as $\neg Y$, are closed under insertions.

Assume by way of contradiction that there is an arena $\mathcal{A} = (V_0, V_1, E, \mathbb{U}, \lambda)$ which is not Y -unambiguous, and take a play α in the intersection of $O_{\text{tail}}^{\mathcal{A}}(Y)$ and $O_{\text{tail}}^{\mathcal{A}}(\neg Y)$. Let $a, b \in \mathbb{N}$ be such that every cycle in $\text{cycles}(\alpha[a, \infty])$ satisfies Y , and every cycle in $\text{cycles}(\alpha[b, \infty])$ satisfies $\neg Y$.

For i, j such that $a \leq i < j$, write $\text{Out}(i, j) = \{\text{cycle}^l(\alpha[a, \infty]) \mid i \leq l \leq j\} \setminus \{\epsilon\}$ to be the set of cycles output by $\text{CycDEC}(\epsilon, \alpha[a, \infty])$ while processing the edges $\alpha_i \dots \alpha_j$, and note that for every $C \in \text{Out}(i, j)$ we have that $\lambda(C) \in Y$. Similarly, for i, j such that $b \leq i < j$, write $\text{Out}^-(i, j) = \{\text{cycle}^l(\alpha[b, \infty]) \mid i \leq l \leq j\} \setminus \{\epsilon\}$ to be the set of cycles output by $\text{CycDEC}(\epsilon, \alpha[b, \infty])$ while processing the edges $\alpha_i \dots \alpha_j$, and note that for every $C \in \text{Out}^-(i, j)$ we have that $\lambda(C) \in \neg Y$.

Since, for every $i \geq 1$, the stack content $\text{stack}^i(\alpha[a, \infty])$ is of length at most $|V| - 1$, there is at least one stack content that appears infinitely often. Thus, let $u_a \in E^*$ be a path of minimal length such that the set $A = \{i \in \mathbb{N} \mid u_a = \text{stack}^i(\alpha[a, \infty])\}$ is infinite. Similarly, let $u_b \in E^*$ be a path of minimal length such that $B = \{i \in \mathbb{N} \mid u_b = \text{stack}^i(\alpha[b, \infty])\}$ is infinite.

We assume w.l.o.g. (otherwise we simply replace A and B by appropriate infinite subsets of themselves until each of the following conditions is satisfied) that \dagger :

- (i) $u_a = \text{stack}^i(\alpha[a, \infty])$ for every $i \in A$, and $u_b = \text{stack}^i(\alpha[b, \infty])$ for every $i \in B$;
- (ii) $\text{end}(\alpha_i) = \text{end}(\alpha_j)$ for all $i, j \in A$ and $\text{end}(\alpha_i) = \text{end}(\alpha_j)$ for all $i, j \in B$ (recall that the nodes on α come from the finite set V);
- (iii) there exists $s_a \in E^*$ such that $s_a = \text{stack}^i(\alpha[a, \infty])$ for all $i \in B$.

Pick indices $b_1, b_2 \in B$ and $a_2, a_3 \in A$ in such a way that $b_1 < a_2 < a_3 < b_2$. [Fig. 6](#) may aid in visualising the current state of affairs. In this figure, $w := \text{end}(\alpha_{b_1}) = \text{end}(u_b) = \text{end}(\alpha_{b_2})$ and $v := \text{end}(\alpha_{a_2}) = \text{end}(u_a) = \text{end}(\alpha_{a_3})$.

Our aim now is to show that the above state of affairs leads to a contradiction, and thus deduce that it can not be that $\alpha \in O_{\text{tail}}^{\mathcal{A}}(Y) \cap O_{\text{tail}}^{\mathcal{A}}(\neg Y)$. The contradiction we will show is that $\lambda(\alpha[b_1 + 1, b_2])$ is both in Y and in its complement $\neg Y$, which is impossible. We will do that by showing how to build $\lambda(\alpha[b_1 + 1, b_2])$ in two different ways: the first by using cyclic permutations and concatenations of strings that are the labels of cycles that satisfy Y , and the second by doing the same but with cycles that satisfy $\neg Y$. Since by the assumption of the proposition Y and $\neg Y$ are closed under these string operations, the contradiction is reached.

We first show that $\alpha[b_1 + 1, b_2]$ (which by $\dagger(ii)$ is a cycle) satisfies $\neg Y$. Intuitively (refer to [Fig. 6](#)), this follows from the fact that all cycles output by the algorithm $\text{CycDEC}(\epsilon, \alpha[b, \infty])$ while processing $\alpha[b_1 + 1, b_2]$ satisfy $\neg Y$, and the fact that before and after processing $\alpha[b_1 + 1, b_2]$ the stack of this algorithm is u_b , and thus by [Lemma 9](#) we have that $\lambda(\alpha[b_1 + 1, b_2])$ is in the insertion closure of the labels of these cycles, and thus also in $\neg Y$ (recall that $\neg Y$ is closed under insertions).

We next show that $\lambda(\alpha[b_1 + 1, b_2]) \in Y$. Observe (refer to [Fig. 6](#)) that $\alpha[b_1 + 1, b_2] = \alpha[b_1 + 1, a_2]\alpha[a_2 + 1, a_3]\alpha[a_3 + 1, b_2]$. Since Y is closed under concatenation and cyclic permutations, to show that $\lambda(\alpha[b_1 + 1, b_2]) \in Y$ it is enough to show that $\lambda(\alpha[a_2 + 1, a_3]) \in Y$ and $\lambda(x) \in Y$, where $x := \alpha[a_3 + 1, b_2]\alpha[b_1 + 1, a_2]$. The fact that $\lambda(\alpha[a_2 + 1, a_3]) \in Y$ follows from a symmetric argument that mimics the one used above to prove that $\alpha[b_1 + 1, b_2]$ satisfies $\neg Y$. It remains to show that $\lambda(x) \in Y$.

To see that $\lambda(x) \in Y$, note that when the algorithm CycDEC($\epsilon, \alpha[a, \infty]$) finishes processing $\alpha[a_3 + 1, b_2]$ it has the same stack content (namely s_a) that it had when it previously started processing $\alpha[b_1 + 1, a_2]$. Thus, if we imagine that after processing $\alpha[a_3 + 1, b_2]$, instead of continuing to process $\alpha[b_2 + 1, \infty]$, we feed the algorithm again with $\alpha[b_1 + 1, a_2]$ (i.e., we let it process the second part of x), we will get (by Lemma 2) that it will behave exactly the same as when it first processed $\alpha[b_1 + 1, a_2]$ as part of the prefix $\alpha[a, a_2]$. We thus obtain that while processing x this way starting with stack u_a , the algorithm ends with stack u_a and outputs only cycles that satisfy Y (recall that all cycles output by CycDEC($\epsilon, \alpha[a, \infty]$) do). Thus, by Lemma 9, we have that $\lambda(x)$ is in the insertion closure of the labels of cycles that satisfy Y , and thus also in Y (recall that Y is closed under insertions). \square

The following Proposition deals with Part 2 in the proof of Theorem 9.

Proposition 3. Fix a Y -unambiguous arena \mathcal{A} . Then for $\sigma \in \{0, 1\}$,

$$WR^\sigma(\mathcal{A}, O_{\text{first}}(Y)) = WR^\sigma(\mathcal{A}, O_{\text{tail}}(Y)).$$

Proof. By Theorem 7 (item 1) it is sufficient to show that if \mathcal{A} is Y -unambiguous then the game $(\mathcal{A}, O_{\text{tail}}(Y))$ is Y -greedy. So, fix a play π in \mathcal{A} . If every cycle in the cycles-decomposition of π satisfies Y then certainly $\pi \in O_{\text{tail}}(Y)$ (just take π itself as the required suffix). On the other hand, if every cycle in the cycles-decomposition of π satisfies $\neg Y$ then for the same reason $\pi \in O_{\text{tail}}(\neg Y)$. However, since \mathcal{A} is Y -unambiguous, $\pi \notin O_{\text{tail}}(Y)$, as required. \square

The following Proposition deals with Part 3.

Proposition 4. Fix a Y -unambiguous arena \mathcal{A} and vertex $z \in V$. Then for $\sigma \in \{0, 1\}$,

$$WR^\sigma(\mathcal{A}, O_{z\text{-first}}(Y)) = WR^\sigma(\mathcal{A}, O_{\text{tail}}(Y)).$$

Proof. Let $\mathcal{A} = (V_0, V_1, E, \mathbb{U}, \lambda)$. We first claim that for every $\sigma \in \{0, 1\}$, we have that $WR^\sigma(\mathcal{A}, O_{z\text{-first}}(Y)) \subseteq WR^\sigma(\mathcal{A}, O_{\text{tail}}(Y))$. To see that the Proposition follows from this claim note that $WR^0(\mathcal{A}, O_{z\text{-first}}(Y))$ and $WR^1(\mathcal{A}, O_{z\text{-first}}(Y))$ partition V since the game $(\mathcal{A}, O_{z\text{-first}}(Y))$, being finitary, is determined (by Lemma 3). Since $WR^0(\mathcal{A}, O_{\text{tail}}(Y))$ and $WR^1(\mathcal{A}, O_{\text{tail}}(Y))$ are disjoint, the containments above must be equalities.

To prove the claim, recall that since \mathcal{A} is Y -unambiguous then $O_{\text{tail}}^{\mathcal{A}}(Y) \cap O_{\text{tail}}^{\mathcal{A}}(\neg Y) = \emptyset$. Hence, a play π in the game $(\mathcal{A}, O_{\text{tail}}(Y))$ is won by Player 0 (resp. Player 1) if π has a suffix π' for which every cycle in $\text{cycles}(\pi')$ is in Y (resp. $\neg Y$). It is thus enough to show that: (†) for every $\sigma \in \{0, 1\}$, and every node v in \mathcal{A} , given a strategy S that is winning from v for Player σ in the game $(\mathcal{A}, O_{z\text{-first}}(Y))$, we can construct a strategy T for Player σ in the game $(\mathcal{A}, O_{\text{tail}}(Y))$, such that every play $\pi \in \text{plays}(T, v)$ has a suffix π' for which every cycle in $\text{cycles}(\pi')$ satisfies Y^σ .

Consider first the case where $z \notin \text{reach}(S, v)$, or that $z \in \text{reach}(S, v)$ but that every path in $\text{plays}(S, v)$ that visits z contains a cycle before the first occurrence of z on the path. Observe that this implies that S is winning from v in the game $(\mathcal{A}, O_{\text{first}}(Y))$. In this case we let $T = S^\circlearrowleft$, where S^\circlearrowleft is the pumping strategy of S (Definition 6). By Lemma 7 we have that, for every $\pi \in \text{plays}(T, v)$, all the cycles in $\text{cycles}(\pi)$ satisfy Y^σ , and thus (†) holds with $\pi' = \pi$.

Consider now the remaining case that $z \in \text{reach}(S, v)$ and that there is a simple path ρ from v to z which is consistent with S . Define a strategy S_z for Player σ in the game $(\mathcal{A}, O_{\text{first}}(Y))$ as follows: for every $u \in V^*V_\sigma$, let $S_z(u) := S(\rho u)$ if u starts in z , and otherwise define it arbitrarily (i.e., $S_z(u) = w$, where w is some node with $(\text{end}(u), w) \in E$). Observe that, for every $\pi \in \text{plays}(S_z, z)$, the play $\rho\pi$ is in $\text{plays}(S, v)$, and since ρ is a simple path that ends in z , Player σ wins the play $\rho\pi$ in the game $(\mathcal{A}, O_{z\text{-first}}(Y))$ iff the first cycle on π satisfies Y^σ . Thus, since S is winning from v , we have that S_z is winning from z in the game $(\mathcal{A}, O_{\text{first}}(Y))$.

We can now construct the desired strategy T for Player σ in the game $(\mathcal{A}, O_{\text{tail}}(Y))$. The strategy T works as follows: as long as a play does not touch the node z the strategy T behaves like the pumping strategy S^\circlearrowleft of S ; however, once (and if) z is reached, T erases its memory and switches to behave like the pumping strategy $(S_z)^\circlearrowleft$ of S_z (starting from z). Recall that we have to show that every play $\pi \in \text{plays}(T, v)$ has a suffix π' for which every cycle in $\text{cycles}(\pi')$ satisfies Y^σ . Informally, if z does not appear on π then π is consistent with S^\circlearrowleft , and by Lemma 7 every cycle in $\text{cycles}(\pi)$ satisfies Y^σ , and we can take $\pi' = \pi$. On the other hand, if z appears on π then $\text{tail}_z(\pi)$ is consistent with $(S_z)^\circlearrowleft$, and thus by Lemma 7 every cycle in $\text{cycles}(\text{tail}_z(\pi))$ satisfies Y^σ , and we can take $\pi' = \text{tail}_z(\pi)$.

Formally, define the strategy T as follows: for every $u \in V^*V_\sigma$,

$$T(u) := \begin{cases} S^\circlearrowleft(u) & \text{if } z \text{ does not appear on } u, \\ (S_z)^\circlearrowleft(\text{tail}_z(u)) & \text{otherwise.} \end{cases}$$

Given $\pi \in \text{plays}(T, v)$, either z appears on π or not. If it does not, then $\pi \in \text{plays}(S^\circlearrowleft, v)$, and by Lemma 7 every cycle C in $\text{cycles}(\pi)$ is the first cycle of some path ρ consistent with S and starting with v , that only uses edges from π . Thus, z does not appear on ρ , and since S is winning from v in the game $(\mathcal{A}, O_{z\text{-first}}(Y))$, we have that C satisfies Y^σ . If z does appear

on π , we argue that $\text{tail}_z(\pi) \in \text{plays}((S_z)^\circlearrowleft, z)$, i.e., that $\text{tail}_z(\pi)$ is consistent with $(S_z)^\circlearrowleft$. Indeed, if $m = |\text{head}(\pi)|$, then for every $j \geq 1$ for which $\text{start}(\text{tail}_z(\pi)_j) \in V_\sigma$ we have:

$$\begin{aligned}\text{end}(\text{tail}_z(\pi)_j) &= T(\pi[1, m+j]) \\ &= (S_z)^\circlearrowleft(\text{tail}_z(\pi[1, m+j])) \\ &= (S_z)^\circlearrowleft(\pi[m+1, m+j]) \\ &= (S_z)^\circlearrowleft((\text{tail}_z(\pi))[1, j]).\end{aligned}$$

By [Lemma 7](#), since $\text{tail}_z(\pi) \in \text{plays}((S_z)^\circlearrowleft, z)$, every cycle in $\text{cycles}(\text{tail}_z(\pi))$ satisfies Y^σ . Overall, in the first case (†) holds with $\pi' = \pi$, and in the second with $\pi' = \text{tail}_z(\pi)$, which completes the proof of the claim. \square

This concludes the proof of [Theorem 9](#).

8. A recipe for proving that a game is memoryless determined

We synthesise the results of the previous section and provide a practical way of deducing uniform memoryless determinacy of many infinite-duration games.

First, we get the following sufficient condition for memoryless determinacy from [Theorems 8 and 9](#).

Theorem 10. *Let Y be a cycle property that is closed under cyclic permutations, and such that both Y and $\neg Y$ are closed under concatenation. Let W be a winning condition. Every Y -greedy game $(\mathcal{A}, O(W))$ is uniform memoryless determined.*

Second, many winning conditions for infinite-duration games (for example parity and mean-payoff) are such that the outcome of a play does not depend on any finite prefix of the play, but only on some “infinite” property of the play. Such winning conditions are usually called *prefix-independent*. Formally:

Definition 9. Say that a winning condition $W \subseteq \mathbb{U}^\omega$ is *prefix-independent* if $c_1c_2\dots \in W$ implies that the suffix $c_ic_{i+1}\dots \in W$ for every $i \in \mathbb{N}$.

In practice, showing whether or not a given W is prefix-independent is usually very easy. The relevance of prefix-independence to our work is captured by the following theorem.

Theorem 11. *Let W be a prefix-independent winning condition, and let Y be a cycle-property that is closed under cyclic permutations. For every arena \mathcal{A} , if the game $(\mathcal{A}, O(W))$ is Y -greedy then it is uniform memoryless determined.*

Proof. By [Theorem 8](#) it is sufficient to prove that every arena \mathcal{A} is Y -resettable. By [Propositions 3 and 4](#) it is thus sufficient to prove that \mathcal{A} is Y -unambiguous. So, assume by way of contradiction there is a play π and indices $a, b \in \mathbb{N}$ be such that every cycle in $\text{cycles}(\pi[a, \infty))$ satisfies Y , and every cycle in $\text{cycles}(\pi[b, \infty))$ satisfies $\neg Y$. Since $(\mathcal{A}, O(W))$ is Y -greedy on \mathcal{A} we get that, in the game $(\mathcal{A}, O(W))$, Player 0 wins $\pi[a, \infty]$, and Player 1 wins $\pi[b, \infty]$. But this is a contradiction to the assumption that W is prefix-independent. \square

Given a winning condition W and a set of arenas of interest \mathcal{C} (in many cases \mathcal{C} is taken to be all arenas), [Theorems 10 and 11](#) suggest the following recipe for proving that the game $(\mathcal{A}, O(W))$ is uniform memoryless determined for every $\mathcal{A} \in \mathcal{C}$.

- Step 1. Finitise the winning condition $W \subseteq \mathbb{U}^\omega$ to get a cycle property $Y \subseteq \mathbb{U}^*$ that is closed under cyclic permutations.
- Step 2. Show that the game $(\mathcal{A}, O(W))$ is Y -greedy for every $\mathcal{A} \in \mathcal{C}$.
- Step 3. Show that either:
 - (a) Both Y and $\neg Y$ are closed under concatenation; or that:
 - (b) W is prefix independent.

We illustrate the use of the recipe with some examples.

Example 1. We will use the recipe to prove that every parity game is memoryless determined. For Step 1, a natural finitisation of the parity condition $W \subset \mathbb{Z}^\omega$ – which says that the largest priority occurring infinitely often is even – is the cycle property cyc-Parity $\subset \mathbb{Z}^*$ which says that the largest priority occurring is even. This property is clearly closed under

cyclic permutations. For Step 2, it is easy to verify that every parity game is cyc-Parity-greedy, and for Step 3, it is immediate that both cyc-Parity and \neg cyc-Parity are closed under concatenation (as it happens, it is also easy to check that W is prefix-independent).

Example 2. We now consider a slightly more subtle application of the recipe. Consider the following game,¹¹ played on an arena \mathcal{A} whose vertices are labelled¹² using the alphabet $\mathbb{U} = \{a, b\}$. The winning condition $W \subset \mathbb{U}^\omega$ consists of those infinite sequences that contain infinitely many a 's and infinitely many b 's. The natural finite version of W is the cycle property $Y \subset \mathbb{U}^*$ consisting of strings which contain at least one a and at least one b , and it is clearly closed under cyclic permutations. To see that W is Y -greedy on \mathcal{A} , observe that if all cycles in $\text{cycles}(\pi)$ satisfy Y then certainly $\pi \in W$. On the other hand, if all cycles in $\text{cycles}(\pi)$ satisfy $\neg Y$ then no edge that appears on such a cycle goes from a node labelled a to a node labelled b . Thus by Lemma 1 (Page 5) π does not satisfy W .

Unfortunately, $\neg Y$ is not closed under concatenation, which prevents us from applying Step 3(a). However, since W is clearly prefix-independent, we can apply Step 3(b) and conclude that $(\mathcal{A}, O(W))$ is memoryless determined.

Example 3. We conclude with a more sophisticated use of the recipe, applied to the initial credit problem of energy games. We show that either there is an initial credit with which Player 0 (the “energy” player) wins, or that for every initial credit Player 1 wins. In both cases, we show that the winner can use a memoryless strategy. A natural finitisation of the energy condition $W_r \subset \mathbb{Z}^\omega$ – which says that at every point along a play, the sum of the initial credit r and the labels of the edges already traversed is not negative – is the cycle property cyc-Energy $\subset \mathbb{Z}^*$ which says that the sum of the numbers is non-negative. This property is clearly closed under cyclic permutations. Given an arena \mathcal{A} , consider the game $(\mathcal{A}, O_{\text{all}}(\text{cyc-Energy}))$. We first claim that if the initial credit is at least $r = -t(|V| - 1)$, where t is the minimum amongst the negative weights of the arena, the winning regions of the energy game and the game $(\mathcal{A}, O_{\text{all}}(\text{cyc-Energy}))$ coincide and winning strategies transfer from the latter to the former. To see that, observe that a play won by Player 1 in $(\mathcal{A}, O_{\text{all}}(\text{cyc-Energy}))$ is also won by her in the energy game since the energy along the play tends to $-\infty$. On the other hand, let π be a play won by Player 0 in $(\mathcal{A}, O_{\text{all}}(\text{cyc-Energy}))$, and consider some prefix π' of it. Recall that, by Lemma 1, at most $|V| - 1$ edges are not on $\text{cycles}(\pi')$, and thus the energy level at the end of π' is at least the initial credit plus $t(|V| - 1)$. Hence, an initial credit of r suffices for π to be winning for Player 0 also in the energy game. It remains to show, using the recipe, that $(\mathcal{A}, O_{\text{all}}(\text{cyc-Energy}))$ is memoryless determined. As noted before (see Example 1 after Definition 5) every $(\mathcal{A}, O_{\text{all}}(Y))$ is Y -greedy, which completes Step 2. Since step 3a is immediate for cyc-Energy we are done.

9. Discussion and future work

The central algorithmic problem for a class of determined graph games is to decide, given an arena and a starting vertex, which of the players has a winning strategy. We have seen a PSPACE upper bound on the complexity of solving games in $\text{FCG}(Y)$ (assuming Y is computable in PSPACE), and that there are very simple cycle properties Y for which the complexity is PSPACE-complete. What about other Y s such as cyc-Parity and cyc-MeanPayoff_v? Our Strategy Transfer result (Theorem 7) implies that the complexity of solving $\text{FCG}(Y)$ is the same as that of solving Y -greedy games. For instance, since parity games are cyc-Parity-greedy, and the complexity of solving them is not known to be in PTIME – except on restricted classes of arenas, e.g., bounded DAG-width ([2]) and bounded trap-depth ([9]) – the same is true of the complexity of solving games from $\text{FCG}(\text{cyc-Parity})$.

On the other hand, since there are cycle properties Y such that some games in $\text{FCG}(Y)$ are memoryless determined and some are not (for instance, take $Y = \text{cyc-EvenLen}$), the following algorithmic problem naturally presents itself: what is the complexity of deciding, given (a finite description of) Y and an arena \mathcal{A} , whether or not the game $(\mathcal{A}, O_{\text{first}}(Y))$ is memoryless determined? We believe that this problem can be addressed using techniques developed in this paper.

Finally, this paper has dealt with qualitative games (i.e., a player either wins or loses). The exact nature of the theory of quantitative first-cycle games over general cycle properties Y is still to be explored. To what extent do our techniques generalise to quantitative games? or to stochastic ones?

Acknowledgments

We thank Erich Grädel for stimulating feedback which led to an improvement of the recipe in Section 8. We thank the referees for their useful suggestions and careful reading. Benjamin Aminof was supported by the Austrian National Research Network S11403-N23 (RISE) of the Austrian Science Fund (FWF) and by the Vienna Science and Technology Fund (WWTF) through grant ICT12-059.

¹¹ This example was kindly brought to our attention by Erich Grädel, and it prompted us to enhance the recipe that was published in the preliminary work [1] to include Step 3(b).

¹² Recall Remark 1 that states that all the results in this paper also apply to vertex labelling.

References

- [1] B. Aminof, S. Rubin, First cycle games, in: Proceedings 2nd International Workshop on Strategic Reasoning, SR 2014, in: EPTCS, vol. 146, 2014, pp. 83–90.
- [2] D. Berwanger, A. Dawar, P. Hunter, S. Kreutzer, Dag-width and parity games, in: B. Durand, W. Thomas (Eds.), Symposium on Theoretical Aspects of Computer Science, STACS 2006, in: Lecture Notes in Computer Science, vol. 3884, Springer, 2006, pp. 524–536.
- [3] A. Bianco, M. Faella, F. Mogavero, A. Murano, Exploring the boundary of half-positionality, Ann. Math. Artif. Intell. 62 (1–2) (2011) 55–77.
- [4] H. Björklund, S. Sandberg, S.G. Vorobyov, Memoryless determinacy of parity and mean payoff games: a simple proof, Theor. Comput. Sci. 310 (1–3) (2004) 365–378.
- [5] K. Chatterjee, L. Doyen, Energy parity games, Theor. Comput. Sci. 458 (2012) 49–60.
- [6] A. Ehrenfeucht, J. Mycielski, Positional strategies for mean payoff games, Int. J. Game Theory 8 (2) (1979) 109–113.
- [7] D. Gale, F.M. Stewart, Infinite games with perfect information, in: H. Kuhn, A. Tucker (Eds.), Contributions to the Theory of Games, Volume II, in: Annals of Mathematics Studies, vol. AM-28, Princeton University Press, 1953, pp. 245–266.
- [8] H. Gimbert, W. Zielonka, Games where you can play optimally without any memory, in: M. Abadi, L. de Alfaro (Eds.), International Conference on Concurrency Theory, CONCUR 2005, in: Lecture Notes in Computer Science, vol. 3653, 2005, pp. 428–442.
- [9] A. Grinshpun, P. Phalitnonkiat, S. Rubin, A. Tarfulea, Alternating traps in Muller and parity games, Theor. Comput. Sci. 521 (2014) 73–91.
- [10] E. Kopczynski, Half-positional determinacy of infinite games, in: International Colloquium on Automata, Languages and Programming, ICALP 2006, 2006, pp. 336–347.
- [11] M. Sipser, Introduction to the Theory of Computation, PWS Publishing Company, 1997.
- [12] U. Zwick, M. Paterson, The complexity of mean payoff games on graphs, Theor. Comput. Sci. 158 (1&2) (1996) 343–359.

Model Checking Parameterised Multi-Token Systems via the Composition Method*

Benjamin Aminof¹ and Sasha Rubin²

¹ Technische Universität Wien, Austria

benj@forsyte.at

² Università di Napoli “Federico II”, Italy

sasha.rubin@unina.it

Abstract. We study the model checking problem of parameterised systems with an arbitrary number of processes, on arbitrary network-graphs, communicating using multiple multi-valued tokens, and specifications from indexed-branching temporal logic. We prove a composition theorem, in the spirit of Feferman-Vaught (1959) and Shelah (1979), and a finiteness theorem, and use these to decide the model checking problem. Our results assume two constraints on the process templates, one of which is the standard fairness assumption introduced in the cornerstone paper of Emerson and Namjoshi (1995). We prove that lifting any of these constraints results in undecidability. The importance of our work is three-fold: i) it demonstrates that the composition method can be fruitfully applied to model checking complex parameterised systems; ii) it identifies the most powerful model, to date, of parameterised systems for which model checking indexed *branching-time* specifications is decidable; iii) it tightly marks the borders of decidability of this model.

1 Introduction

Many concurrent systems consist of identical processes running in parallel, such as peer-to-peer systems, sensor networks, multi-agent systems, etc. [14,27,29]. Model checking is a successful technique for establishing correctness of such systems: model a system as the product transition system \mathbf{P}^G , where \mathbf{P} is a transition system representing the process, and G is a network-graph describing the communication lines [9]. If the number of processes is not known, or too large for model-checking tools, it is appropriate to express correctness as a parameterised model checking (PMC) problem: decide if a given temporal logic specification holds irrespective of the number of processes [8,22]. That is, for a fixed infinite set \mathcal{G} of network-graphs (e.g., \mathcal{G} may be the set of all ring network-graphs), decide, given process \mathbf{P} and specification φ if $\mathbf{P}^G \models \varphi$ for all $G \in \mathcal{G}$. Not surprisingly, PMC is a hard problem, i.e., even for a given \mathbf{P} , PMC consists of

* Benjamin Aminof is supported by the Vienna Science and Technology Fund (WWTF) through grant ICT12-059. Sasha Rubin is a Marie Curie fellow of the Istituto Nazionale di Alta Matematica.

model-checking infinitely many systems; in other words, it can be thought of as model checking infinite-state systems [15,23]. It quickly becomes undecidable [8], even if the participating processes are finite-state [32], and even if they do not communicate with each other at all [25]. Thus, much work has focused on proving decidability for restricted systems, i.e., by limiting both the communication mechanism and the specification logic [20,17,13,1,2,3,10,6,7].

We consider specifications in indexed branching temporal logic without the “next-time” operator \mathbf{X} (formulas without \mathbf{X} are stuttering-insensitive, and are thus natural for specifying asynchronous concurrent systems [18]). More specifically, we use formulas of prenex indexed- $\text{CTL}_d^*\backslash\mathbf{X}$ (CTL^* without \mathbf{X} in which there at most $d \in \mathbb{N}$ nested path quantifiers). These are formulas of the form $\forall x_1 \exists x_2 \dots \forall x_k \phi$ where the variables x_i vary over processes, and ϕ is a $\text{CTL}_d^*\backslash\mathbf{X}$ formula where atomic propositions are paired with the index variables x_i [30]. This specification language allows one to express many natural properties, e.g. mutual-exclusion. Non-prenex temporal logic is so powerful that its parameterised model checking is undecidable already for indexed $\text{LTL}\backslash\mathbf{X}$ specifications, even for non-communicating processes [25]. We consider systems with an arbitrary number of processes, on arbitrary network-graphs, communicating using multiple multi-valued tokens. Such systems arise in various contexts: multiple tokens are a means to resolve conflicts over multiple shared resources such as in the drinking philosophers problem [12], they can represent mobile finite-state agents [29,4,5], and tokens are used in self-stabilisation algorithms [24]. We further allow the edges of the network-graph to carry directions, called local port-numberings, along which the processes may send and receive tokens. Such network-graphs are typical in the distributed computing literature, for instance in mobile finite-state agents, e.g., [14,27,26]. Note that even slightly more powerful communication primitives such as pairwise-rendezvous have undecidable PMC for expressive logics such as prenex indexed $\text{CTL}^*\backslash\mathbf{X}$ [3].

The Compositional Method for Parameterised Model-Checking. Composition theorems, pioneered in the seminal work of Feferman and Vaught [21] and Shelah [31], are tools that reduce reasoning about compound structures to reasoning about their component parts. Unfortunately, composition theorems for product systems are not easy to come by [28]. Nonetheless, we successfully apply the composition method to multi-token systems and prenex indexed- $\text{CTL}_d^*\backslash\mathbf{X}$ specification languages. Our composition result (Theorem 3) states, roughly, that if two processes \mathbf{X}, \mathbf{Y} are bisimilar, and if two network-graphs G, H with k visible vertices \bar{g}, \bar{h} (i.e., vertices that formulas can talk about) are $\text{CTL}_d^*\backslash\mathbf{X}$ -equivalent, then the product systems $\mathbf{X}^G, \mathbf{Y}^H$ with visible processes at \bar{g}, \bar{h} are $\text{CTL}_d^*\backslash\mathbf{X}$ -equivalent. We complement this with a finiteness result (Theorem 5) that states, roughly, that for every $d, k \in \mathbb{N}$, there are only finitely many $\text{CTL}_d^*\backslash\mathbf{X}$ -types of network-graphs G with k visible vertices (even though, over all graphs, there are infinitely many logically inequivalent $\text{CTL}_d^*\backslash\mathbf{X}$ formulas, already for $d = 1$). Combining the composition and finiteness we reduce reasoning about \mathbf{P}^G for all $G \in \mathcal{G}$ to reasoning about finitely many $G \in \mathcal{G}$, and thus decide the PMC.

Our systems employ two fairness conditions: the standard assumption (introduced in [18]) that processes that make infinitely many transitions must make infinitely many token passing transitions; and the assumption that from every state from which a process can send (resp. receive) a token, it can also reach a state in which it can send (resp. receive) the token in any other given direction and value. We show that if either of the fairness conditions is removed PMC becomes undecidable; furthermore, it remains undecidable even if other very restrictive assumptions are added. It is notable that until now it was not known if the standard fairness assumption was necessary for decidability. Thus, our results answer this question in the affirmative. Due to space constraints, some proofs are only sketched or omitted.

2 Definitions

A *labeled transition system (LTS)* is a tuple $\langle \text{AP}, \Sigma, Q, Q_0, \delta, \lambda \rangle$ where AP is a finite set of *atomic propositions* (also called *atoms*), Σ is a finite set of *actions*, Q is a finite set of *states*, $Q_0 \subseteq Q$ is a set of *initial states*, $\delta \subseteq Q \times \Sigma \times Q$ is a *transition relation*, and $\lambda : Q \rightarrow 2^{\text{AP}}$ is a *labeling function*. We write $q \xrightarrow{\sigma} q'$ if $(q, \sigma, q') \in \delta$, and write $q \rightarrow q'$ if $(q, \sigma, q') \in \delta$ for some $\sigma \in \Sigma$. An LTS is *total* if for every $q \in Q$ there exists $q' \in Q$ such that $q \rightarrow q'$. A *transition system (TS)* is a tuple $\langle \text{AP}, Q, Q_0, \delta, \lambda \rangle$ like an LTS, except that $\delta \subseteq Q \times Q$. A *path* of an LTS is a finite string $q_0 q_1 \dots q_n \in Q^+$ or an infinite string $q_0 q_1 \dots \in Q^\omega$ such that $q_i \rightarrow q_{i+1}$ for all i . An *edge-path* of an LTS is a (finite or infinite) sequence of transitions $(q_0, \sigma_0, q_1)(q_1, \sigma_1, q_2) \dots$ of δ . Every edge-path $(q_0, \sigma_0, q_1)(q_1, \sigma_1, q_2) \dots$ induces the path $q_0 q_1 q_2 \dots$. A path is *simple* if no vertex repeats, and it is a *simple cycle* if the (only) two equal vertices are the first and last. An edge-path is *simple* (resp. *simple cycle*) if the induced path is. A *run* of an LTS is a maximal path starting in an initial state. An LTS can be translated into a TS by simply removing the actions from transitions. We will implicitly use this translation.

System Model. Informally, a token-passing system is an LTS obtained by taking some finite edge-labeled graph (called a *topology* or *network-graph*), placing one process at each of its vertices, and having all processes execute the same code (given in the form of a finite-state *process template*). Processes synchronize by sending one of finitely many tokens along the edges of the topology, which are labeled with a send direction, a receive direction, and a token-value.³ In the most general model, processes can choose the direction to send the token, from which direction to receive a token, and the value of the token. In case there is more than one possible recipient for a token, one is chosen nondeterministically

In what follows, we use a finite non-empty set of *token values* Σ_{val} , finite disjoint non-empty sets Σ_{snd} of *send directions* and Σ_{rcv} of *receive directions*, and an integer $T > 0$ (the number of tokens in the system). Since these data are usually fixed, we do not mention them if they are clear from the context.

³ The direction-labels on the edges (also called a local orientation) represent network port numbers [14,27]. All of our results also hold for the case that each edge has a single direction-label that combines send and receive directions, e.g., “clockwise”.

Process Template. Fix a countable set \mathcal{AP} of atomic propositions for use by all process templates. We assume that \mathcal{AP} also contains, for every integer $i \geq 0$, the special proposition tok^i . A *process template* (w.r.t. $T \in \mathbb{N}$, Σ_{val} , Σ_{snd} , Σ_{rcv}) is a total LTS $\mathbf{P} = \langle \mathcal{AP}_{\text{pr}}, \Sigma_{\text{pr}}, Q, Q_0, \delta, \lambda \rangle$ such that: (i) $\mathcal{AP}_{\text{pr}} \subset \mathcal{AP}$ is a finite set containing tok^i for $0 \leq i \leq T$; (ii) for every $q \in Q$ there is exactly one i such that $\text{tok}^i \in \lambda(q)$ (and we say that q has i tokens); (iii) $\Sigma_{\text{pr}} = \{\text{int}\} \cup [(\Sigma_{\text{snd}} \cup \Sigma_{\text{rcv}}) \times \Sigma_{\text{val}}]$; (iv) $Q_0 = \{\iota_T, \iota_0\}$ where ι_T has T tokens, and ι_0 has 0 tokens; (v) For every transition $q \xrightarrow{\sigma} q'$: if $\sigma \in \Sigma_{\text{snd}} \times \Sigma_{\text{val}}$ then $\exists i > 0$ such that q has i tokens and q' has $(i - 1)$ tokens; if $\sigma \in \Sigma_{\text{rcv}} \times \Sigma_{\text{val}}$ then $\exists i < T$ such that q has i tokens and q' has $(i + 1)$ tokens; and if $\sigma = \text{int}$ then q and q' have the same number of tokens.

Notation. We say that the *initial states* of two templates \mathbf{X}, \mathbf{Y} are *bisimilar* if, writing $\iota_0^{\mathbf{Z}}, \iota_T^{\mathbf{Z}}$ for the initial states of template \mathbf{Z} , we have that $\iota_{\epsilon}^{\mathbf{X}} \sim \iota_{\epsilon}^{\mathbf{Y}}$ for $\epsilon \in \{0, T\}$, where \sim is a bisimulation relation between \mathbf{X} and \mathbf{Y} . The elements of Q are called *local states* and the transitions in δ are called *local transitions*. A transition (q, σ, q') is called a *local send transition* if $\sigma \in \Sigma_{\text{snd}} \times \Sigma_{\text{val}}$; it is called a *local receive transition* if $\sigma \in \Sigma_{\text{rcv}} \times \Sigma_{\text{val}}$; and it is called a *local internal transition* if $\sigma = \text{int}$. The local send/receive transitions are collectively known as *local token-passing transitions*. A local state q for which there exists a local send-transition (resp. receive-transition) $(q, (d, m), q')$ is called *ready to send* (resp. *receive*) *in direction d and value m*; it is also called *ready to send* (resp. *receive*) *in direction d, ready to send* (resp. *receive*) *value m*, or simply *ready to send* (resp. *receive*).

Fairness Notions. A template \mathbf{P} is *fair* if every infinite path $q_1 q_2 \dots$ in \mathbf{P} satisfies that for infinitely many i the transition from q_i to q_{i+1} is a local token passing transition [18]. Other restrictions that we consider involve treating different directions and/or different token values in an unbiased way, and thus “fairly”. Formally, a state q of \mathbf{P} having i tokens that is ready to send (resp. receive) is called an *i-sending* (resp. *i-receiving*) state. A path in \mathbf{P} is an *i-path* if it only mentions states having i tokens. A template \mathbf{P} is *direction/value-fair* if for every $d \in \Sigma_{\text{rcv}}$, $e \in \Sigma_{\text{snd}}$ and $m \in \Sigma_{\text{val}}$, for every *i-receiving* (resp. *i-sending*) state q there is a finite *i-path* from q ending in a state that is ready to receive (resp. send) in direction d (resp. from direction e) and value m . We denote by \mathcal{P}^{FDV} the set of fair and direction/value-fair process-templates; and by \mathcal{P}^{FD} the set of fair, direction-fair, and valueless (i.e., with $|\Sigma_{\text{val}}| = 1$) process templates. As noted in the introduction, the undecidability results (Section 4) show that the limitations of \mathcal{P}^{FDV} are, in a strong sense, minimal limitations one can impose and still obtain a decidable parameterized model checking problem.

Topology/Network-Graph. LTS $G = \langle \emptyset, \Sigma_{\text{snd}} \times \Sigma_{\text{rcv}}, V, \{\text{init}\}, E, \lambda \rangle$ is a *topology* (w.r.t. Σ_{snd} , Σ_{rcv}) if: (i) $V = [n]$ for some $n \in \mathbb{N}$ is a set of *vertices* (or *process indices*); (ii) $\text{init} \in V$ is an *initial vertex*; (iii) $E \subseteq V \times (\Sigma_{\text{snd}} \times \Sigma_{\text{rcv}}) \times V$ is called the *edge relation*, (iv) and λ is the constant function $\lambda(v) = \emptyset$. We abbreviate and write $G = \langle V, E, \text{init} \rangle$ or $G = \langle V_G, E_G, \text{init}_G \rangle$. The *underlying graph* of G has vertex set V and edge (v, w) iff $\exists d, e. (v, (d, e), w) \in E$. We assume that the underlying graph is irreflexive, contains no vertices without outgoing edges, and that every vertex $v \in V$ is reachable from init . These are natural assumptions since paths in the topology represent the paths along which the tokens can move.

Parameterized Topology \mathcal{G} . Let \mathcal{G} denote a countable set of topologies. For example, the set of all pipelines, or the set of all rings.



Fig. 1: example pipeline topology: + signifies a $(\text{snd}_E, \text{rcv}_W)$ label, and - $(\text{snd}_W, \text{rcv}_E)$.

Token-Passing System. Given a process template $\mathbf{P} = \langle \text{AP}_{\text{pr}}, \Sigma_{\text{pr}}, Q, Q_0, \delta, \lambda \rangle$ and a topology $G = \langle V, E, \text{init} \rangle$, we define the *token-passing system* (or TPS for short) to be the LTS $\mathbf{P}^G = \langle \text{AP}_{\text{sys}}, \Sigma_{\text{sys}}, S, S_0, \Delta, \Lambda \rangle$. Informally, the system \mathbf{P}^G can be thought of as the interleaving parallel composition of \mathbf{P} over G . The tokens start with process init. Time is discrete: at each step either exactly one process makes a local internal transition; or exactly two processes, say at vertices v, w , simultaneously make local token-passing transitions as the process at v sends a token in direction d with value m , and the one at w receives the token from direction e with value m . Such a transition can occur only if $(v, (d, e), w) \in E$.

Formally: (1) $\text{AP}_{\text{sys}} := \text{AP}_{\text{pr}} \times V$ is the set of *indexed atomic propositions* (because it is standard notation, we sometimes write p_i instead of (p, i)); (2) $\Sigma_{\text{sys}} := \{\text{int}\} \cup (\Sigma_{\text{snd}} \times \Sigma_{\text{rcv}} \times \Sigma_{\text{val}})$ is the set of actions; (3) The set of *global states* is $S := Q^V$, i.e., all functions from V to Q (informally, if $s \in Q^V$ is a global state then $s(i)$ denotes the local state of the process with index i); (4) The set of *global initial states* S_0 consists of the unique global state $s \in Q_0^V$ such that $s(\text{init}) = \iota_T$, and for all $i \neq \text{init}$, $s(i) = \iota_0$; (5) The labeling $\Lambda(s) \subset \text{AP}_{\text{sys}}$ for $s \in S$ is defined as follows: $p_i \in \Lambda(s)$ if and only if $p \in \lambda(s(i))$, for $p \in \text{AP}_{\text{pr}}$ and $i \in V$ (informally, p_i is true at s if and only if p is true at the corresponding local state of the process with index i);

(6) The *global transition relation* $\Delta \subseteq S \times \Sigma_{\text{sys}} \times S$ consists of global internal transitions and global token-passing transitions (collectively called *global transitions*), defined as follows: (6a) The *global internal transitions* are elements of the form (s, int, s') for which there exists a process index $v \in V$ such that $s(v) \xrightarrow{\text{int}} s'(v)$ is a local internal transition of \mathbf{P} , and for all $w \in V \setminus \{v\}$, $s(w) = s'(w)$. Such a global transition is said to *involve* v . (6b) The *global token-passing transitions* are elements of the form $(s, (d, e, m), s')$ for which there exist process indices $v, w \in V$ such that: $(v, (d, e), w) \in E$; there is a local send transition $s(v) \xrightarrow{(d, m)} s'(v)$ and a local receive transition $s(w) \xrightarrow{(e, m)} s'(w)$ of \mathbf{P} ; and for every $u \in V \setminus \{v, w\}$, $s'(u) = s(u)$. Such a transition is said to have v *sending in direction d an m-valued token to w from direction e*.

Observe that the definition above specifies that at the beginning all tokens are at the initial state of the topology. This is not a real restriction since a system which allows the tokens to start already distributed in a nondeterministic way among multiple vertices can be simulated by adding to the topology a new initial state that starts with all the tokens and from which they are later distributed. One can even accommodate many fixed initial distributions by modifying the specification formula to remove from consideration unwanted distributions.

For a global state s , let $\text{tokens}(s)$ be the set of v such that $s(v)$ has one or more tokens. If $T = 1$, we define $\text{tokens}(s)$ to be the vertex that has the token.

Specification Language. For the syntax and semantics of CTL^* see [9]. In this work, TL denotes a syntactic fragment of CTL^* , such as $\text{CTL}^*\setminus\text{X}$ (i.e., CTL^* without the “next” operator), or the fragment $\text{CTL}_d^*\setminus\text{X}$ of $\text{CTL}^*\setminus\text{X}$ in which the nesting-depth of the path quantifiers E, A is at most $d \in \mathbb{N}_0$ (see [30]).

A *partition* of an infinite path $\pi = \pi_1\pi_2\dots$ is an infinite sequence B_1, B_2, \dots of finite intervals of \mathbb{N} such that there exist integers $m_1 < m_2 < \dots$ with $m_1 = 1$ and for all $i \in \mathbb{N}$, $B_i = [m_i, m_{i+1} - 1]$. The intervals B_i are called *blocks*.

Definition 1. [30] For TSSs $M = \langle \text{AP}, S, S_0, \Delta, \Lambda \rangle$, $M' = \langle \text{AP}, S', S'_0, \Delta', \Lambda' \rangle$ (over the same set of atomic propositions AP), and non-negative integer d , define relations $\equiv_d \subseteq S \times S'$ as follows: (i) $s \equiv_0 s'$ if $\Lambda(s) = \Lambda'(s')$; and (ii) $s \equiv_{d+1} s'$ if for every infinite path π in M from s there exists an infinite path π' in M' from s' (and vice versa) and a partition $B_1B_2\dots$ of π and a partition $B'_1B'_2\dots$ of π' such that for every $i \in \mathbb{N}$ and every $b \in B_i, b' \in B'_i$ we have that $\pi_b \equiv_d \pi'_{b'}$.

In case we need to stress the LTSs, we write $\equiv_d^{M,M'}$ instead of \equiv_d . Say that M is *TL-equivalent* to M' , denoted by $M \equiv_{\text{TL}} M'$, if they agree on all TL formulas, i.e., for every TL formula φ over AP it holds that $M \models \varphi$ iff $M' \models \varphi$. The next proposition characterizes $\text{CTL}_d^*\setminus\text{X}$ -equivalence, denoted $\equiv_{\text{CTL}_d^*\setminus\text{X}}$.

Proposition 1. [30] For every integer d , TS M with a single initial state s , and TS M' with a single initial state s' : $M \equiv_{\text{CTL}_d^*\setminus\text{X}} M'$ if and only if $s \equiv_d s'$.

Indexed Temporal Logics (ITLs) were introduced in [11,19,18] to model specifications of systems with multiple processes. ITL formulas are built from TL formulas by adding the ability to quantify over process indices using the universal and existential process quantifiers $\forall x_{\text{cond}}$ and $\exists x_{\text{cond}}$ (generally written as Qx). Accordingly, the atoms are $\text{AP} \times \text{Vars}$, where $\text{Vars} = \{x, y, z, \dots\}$ is some fixed infinite set of index variables (we write p_x instead of $(p, x) \in \text{AP} \times \text{Vars}$). For example, the formula $\forall x \forall y_{x \neq y}. \text{A} \neg \text{F} c_x \wedge c_y$ specifies mutual exclusion, i.e., that it is never the case that two different processes simultaneously satisfy atom c . Syntactically, *Indexed-CTL** formulas are formed by adding the following to the syntax of CTL^* formulas over atomic propositions $\text{AP} \times \text{Vars}$: if φ is an indexed- CTL^* state (resp. path) formula then so are the formulas $\forall x_{\text{cond}}.\varphi$ and $\exists x_{\text{cond}}.\varphi$, where $x, y \in \text{Vars}$, and cond is Boolean combination over predicates of the form true , $(x, y) \in E$, $(y, x) \in E$, and $x = y$.

Semantics. Indexed- CTL^* formulas over variables Vars and atomic propositions AP_{pr} are interpreted over a token-passing system \mathbf{P}^G , where \mathbf{P} has atomic propositions AP_{pr} , and $G = \langle V, E, \text{init} \rangle$. A *valuation* is a function $e : \text{Vars} \rightarrow V_G$. An x -variant of e is a valuation e' with $e'(y) = e(y)$ for all $y \in \text{Vars} \setminus x$. First we inductively define what it means for *valuation e to satisfy cond*, written $e \models \text{cond}$: $e \models \text{true}$ (for all e); $e \models x = y$ iff $e(x) = e(y)$; $e \models (x, y) \in E$ iff $(e(x), e(y)) \in E$; $e \models \neg \text{cond}$ iff $e \not\models \text{cond}$; $e \models \text{cond} \wedge \text{cond}'$ iff $e \models \text{cond}$ and $e \models \text{cond}'$.

For a TPS $\mathbf{P}^G = \langle \text{AP}_{\text{sys}}, \Sigma_{\text{sys}}, S, S_0, \Delta, \Lambda \rangle$, a global state s , a state formula φ , and a valuation e , define $(\mathbf{P}^G, s) \models \varphi[e]$ inductively:

- $(\mathbf{P}^G, s) \models p_x[e]$ iff $p_{e(x)} \in \Lambda(s)$,
- $(\mathbf{P}^G, s) \models \mathsf{E} \psi[e]$ iff $(\mathbf{P}^G, \pi) \models \psi[e]$ for some infinite path π from s in \mathbf{P}^G ,
- $(\mathbf{P}^G, s) \models \forall x_{cond}.\varphi[e]$ (resp. $(\mathbf{P}^G, s) \models \exists x_{cond}.\varphi[e]$) iff for all (resp. for some) x -variants e' of e that satisfy $cond$, it holds that $(\mathbf{P}^G, s) \models \varphi[e']$,
- $(\mathbf{P}^G, s) \models \varphi \wedge \varphi'[e]$ iff $(\mathbf{P}^G, s) \models \varphi[e]$ and $(\mathbf{P}^G, s) \models \varphi'[e]$, and
- $(\mathbf{P}^G, s) \models \neg \varphi[e]$ iff it is not the case that $(\mathbf{P}^G, s) \models \varphi[e]$.

Path formulas are interpreted similarly, but over (\mathbf{P}^G, π) , where π is an infinite path. An indexed CTL* formula is a *sentence* if every atom is in the scope of a process quantifier. Let φ be an indexed-CTL* state formula. For a valuation e , define $\mathbf{P}^G \models \varphi[e]$ if $(\mathbf{P}^G, s_0) \models \varphi[e]$, where s_0 is the initial state of \mathbf{P}^G . If φ is also a sentence, define $\mathbf{P}^G \models \varphi$ if for all valuations e (equivalently, for some valuation) it holds that $(\mathbf{P}^G, s_0) \models \varphi[e]$. Similarly, define $(\mathbf{P}^G, s) \models \varphi$ iff for all valuations (equivalently, for some valuation) $e : \mathsf{Vars} \rightarrow V_G$ it holds that $(\mathbf{P}^G, s) \models \varphi[e]$. We use the usual shorthands, e.g., $\forall x.\varphi$ is shorthand for $\forall x_{\text{true}}.\varphi$.

Prenex indexed-TL is a syntactic fragment of indexed-TL in which all the processes' index quantifiers are at the front of the formula, e.g., prenex indexed CTL* $\setminus X$ consists of formulas of the form $(Q_1 x_1) \dots (Q_k x_k) \varphi$ where φ is a CTL* $\setminus X$ formula over atoms $\mathsf{AP} \times \{x_1, \dots, x_k\}$, and the $Q_i x_i$ s are index quantifiers. Such formulas with k quantifiers are called *k-indexed*, collectively written $\{\forall, \exists\}^k\text{-TL}$. The union of $\{\forall, \exists\}^k\text{-TL}$ for $k \in \mathbb{N}$ is written $\{\forall, \exists\}^*\text{-TL}$ and called (full) prenex indexed TL. The remainder of this paper deals with prenex indexed-CTL*_d $\setminus X$.

Parameterized Model Checking Problem $\mathbf{PMCP}_G(\mathcal{P}, \mathcal{F})$. The *parameterized model checking (PMC) problem* is to decide, given $\mathbf{P} \in \mathcal{P}$ and $\varphi \in \mathcal{F}$, whether or not for all $G \in \mathcal{G}$, $\mathbf{P}^G \models \varphi$. Here \mathcal{P} is a set of process templates, and \mathcal{F} is a set of ITL formulas.

Cutoffs and Decidability. A *cutoff* for $\mathbf{PMCP}_G(\mathcal{P}, \mathcal{F})$ is a natural number c such that for every $\mathbf{P} \in \mathcal{P}$ and $\varphi \in \mathcal{F}$, if $\mathbf{P}^G \models \varphi$ for all $G \in \mathcal{G}$ with $|V_G| \leq c$ then $\mathbf{P}^G \models \varphi$ for all $G \in \mathcal{G}$. Note: if $\mathbf{PMCP}_G(\mathcal{P}, \mathcal{F})$ has a cutoff, then it is decidable. Note that the existence of a cutoff only implies the existence of a decision procedure. For instance, the statement “for every $k \in \mathbb{N}$, $\mathbf{PMCP}_G(\mathcal{P}, \{\forall, \exists\}^k\text{-TL})$ has a cutoff” does not imply, a priori, that $\mathbf{PMCP}_G(\mathcal{P}, \{\forall, \exists\}^*\text{-TL})$ is decidable.

3 Decidability Results

In this section we prove that token-passing systems have decidable PMC problem for specifications from k -indexed CTL*_d $\setminus X$ for fair and direction/value-fair process templates. We begin with some definitions.

Notation. A *k-tuple* over V_G , written \bar{g} , denotes a tuple (g_1, \dots, g_k) of elements of V_G . We write $v \in \bar{g}$ if $v = g_i$ for some i . Given a valuation $e : \mathsf{Vars} \rightarrow V_G$, the relevant part of e for a CTL*_d $\setminus X$ formula with k free variables (w.l.o.g. called x_1, \dots, x_k) can be described by a k -tuple \bar{g} over V_G (with $g_i = e(x_i)$ for $1 \leq i \leq k$).

The restriction $\mathbf{P}^G|\bar{g}$. Fix process template \mathbf{P} , topology G , and nodes $\bar{g} \in V_G^k$. Define the *restriction* of $\mathbf{P}^G = \langle \mathsf{AP}_{\text{sys}}, S, S_0, \Delta, \Lambda \rangle$ onto \bar{g} , written $\mathbf{P}^G|\bar{g}$, as the LTS $(\mathsf{AP}_{\text{@}}, S, S_0, \Delta, L)$ over atomic propositions $\mathsf{AP}_{\text{@}} = \{p@i : p \in \mathsf{AP}_{\text{pr}}, i \in [k]\}$,

where for all $s \in S$ the labeling $L(s)$ is defined as follows: $L(s) := \{p@i : p_{g_i} \in \Lambda(s), i \in [k]\}$. Informally, $\mathbf{P}^G|\bar{g}$ is the LTS \mathbf{P}^G with a modified labeling that, for every $g_i \in \bar{g}$, replaces the indexed atom p_{g_i} by the atom $p@i$ (i.e., process indices are replaced by their *positions* in \bar{g}); all other atoms are removed. Intuitively, $p@i$ means that the atom $p \in \text{AP}_{\text{pr}}$ holds in the process with index (i.e., at the vertex) g_i . Note that \mathbf{P}^G and $\mathbf{P}^G|\bar{g}$ only differ in their labelling. It is not hard to see that given a k -indexed formula $\theta := Q_1x_1 \dots Q_kx_k. \varphi$, the truth value of φ in \mathbf{P}^G , with respect to a valuation for x_1, \dots, x_k described by a k -tuple \bar{g} , can be deduced by reasoning instead on $\mathbf{P}^G|\bar{g}$ (since for this evaluation φ only “sees” the atomic propositions of processes in vertices in \bar{g}).

The valuation TS $G[\bar{g}]$. The idea is to annotate the topology G by atoms that allow logical formulae to talk about the movement of tokens in and out of vertices in \bar{g} . In order to capture the directions involved in such movements, we insert new nodes in the middle of any edge of G that is incident with a vertex in \bar{g} . Thus, $G[\bar{g}]$ is a TS formed as follows: i) the atoms true at v are the positions that v appears in \bar{g} , if any; ii) split each edge labeled (d, e) involving (one or two) vertices from \bar{g} by inserting a state whose atoms label the directions to or from the vertices from \bar{g} that are involved; iii) remove all edge labels.

Formally, let $G = \langle V, E, \text{init} \rangle$ be a topology, and let \bar{g} be a k -tuple over V . Define the *valuation TS* $G[\bar{g}]$ as the TS $\langle \text{AP}, Q, Q_0, \delta, \lambda \rangle$ where

- $\text{AP} = [k] \cup \Sigma_{\text{snd}} \cup \Sigma_{\text{rcv}}$,
- $Q = V \cup \{[v, d, e, w] \mid (v, (d, e), w) \in E \text{ and either } v \in \bar{g} \text{ or } w \in \bar{g}\}$,
- $Q_0 = \{\text{init}\}$,
- $\delta \subset Q \times Q$ is the union of $\{(v, v') : \exists d, e. (v, (d, e), v') \in E, v \notin \bar{g} \wedge v' \notin \bar{g}\}$ and $\{(v, [v, d, e, w]) : [v, d, e, w] \in Q\}$ and $\{([v, d, e, w], w) : [v, d, e, w] \in Q\}$.
- $\lambda(v) := \{i \in [k] : v = g_i\}$ (for $v \in V$); and $\lambda([v, d, e, w])$ is $\{d\}$ if $v \in \bar{g}, w \notin \bar{g}$, is $\{e\}$ if $w \in \bar{g}, v \notin \bar{g}$, and is $\{d, e\}$ if $v \in \bar{g}, w \in \bar{g}$.

Since Σ_{snd} and Σ_{rcv} are disjoint, the label of (v, d, e, w) determines which of v and w is in \bar{g} . A valuation TS $G[\bar{g}]$ with $|\bar{g}| = k$ is called a k -*valuation TS*. Every edge-path $\xi \in G$ naturally induces a path $\text{map}(\xi)$ in $G[\bar{g}]$. Observe that $\text{map}(\xi)$ starts in a node of V_G , and if ξ is finite also ends in a node of V_G . Formally: $\text{map}((v, \sigma, v'))$ is defined to be vv' if $v \notin \bar{g}$ and $v' \notin \bar{g}$, and $v \cdot [v, \sigma, v'] \cdot v'$ otherwise; and $\text{map}(\xi \cdot (v, \sigma, v'))$ is defined to be $\text{map}(\xi) \cdot v'$ if $v \notin \bar{g}$ and $v' \notin \bar{g}$, and is $\text{map}(\xi) \cdot [v, \sigma, v'] \cdot v'$, otherwise. Note that for a path ρ in $G[\bar{g}]$ that begins in a node of V_G (and, if ρ is finite, also ends in V_G), the set $\text{map}^{-1}(\rho)$ is non-empty.

We can now define the COMPOSITION and FINITENESS properties.

- **COMPOSITION** Property for $\langle \text{TL}, \mathcal{P}, \mathcal{G} \rangle$: For every $k \in \mathbb{N}$, processes $\mathbf{X}, \mathbf{Y} \in \mathcal{P}$, topologies $G, H \in \mathcal{G}$, and k -tuples $\bar{g} \in V_G^k, \bar{h} \in V_H^k$: if $G[\bar{g}] \equiv_{\text{TL}} H[\bar{h}]$ and the initial states of \mathbf{X} and \mathbf{Y} are bisimilar, then $\mathbf{X}^G|\bar{g} \equiv_{\text{TL}} \mathbf{Y}^H|\bar{h}$. In words, the COMPOSITION property states that if the initial states of \mathbf{X} and \mathbf{Y} are bisimilar then one can deduce the logical equivalence of the restrictions $\mathbf{X}^G|\bar{g}, \mathbf{Y}^H|\bar{h}$ from the logical equivalence of the valuation TSs $G[\bar{g}], H[\bar{h}]$.
- **FINITENESS** Property for $\langle \text{TL}, \mathcal{G} \rangle$: For every $k \in \mathbb{N}$, the set $\mathcal{M} := \{G[\bar{g}] : G \in \mathcal{G}, \bar{g} \in V_G^k\}$ has only finitely many \equiv_{TL} equivalence classes.

Later in this section we will prove the COMPOSITION and FINITENESS properties with $\text{TL} = \text{CTL}_d^* \setminus X$ (for fixed $d \in \mathbb{N}$), $\mathcal{P} = \mathcal{P}^{\text{FD}}$. Note that if instead of using valuation TSs one uses arbitrary TSs then the finiteness property does not hold even for $\text{TL} = \text{CTL}_1^* \setminus X$.⁴ Thus, the proof of the finiteness property for $\text{CTL}_d^* \setminus X$ must, and does, exploit properties of valuation TSs; in particular, the fact that the number of atoms is bounded and no atom is true in more than one state of $G[\bar{g}]$ in every path between two vertices in \bar{g} .

We now state the main theorem of this section.

Theorem 1. $\text{PMCP}_{\mathcal{G}}(\mathcal{P}^{\text{FDV}}, \{\forall, \exists\}^k\text{-}\text{CTL}_d^* \setminus X)$ is decidable for every parameterised topology \mathcal{G} and every $d, k \in \mathbb{N}$.

The proof is in two steps. First, one removes the token values by encoding them in the directions. Thus, in the statement of Theorem 1, we may replace \mathcal{P}^{FDV} by \mathcal{P}^{FD} . In step two, we show that (for every \mathcal{G}, k, d) the PMC problem has a cutoff using the composition method, following the recipe from [2]:

Theorem 2. If $\langle \text{TL}, \mathcal{P}, \mathcal{G} \rangle$ has the COMPOSITION property and $\langle \text{TL}, \mathcal{G} \rangle$ has the FINITENESS property, then for all $k \in \mathbb{N}$, $\text{PMCP}_{\mathcal{G}}(\mathcal{P}, \{\forall, \exists\}^k\text{-}\text{TL})$ has a cutoff.

Proof (sketch). The truth value of a $\{\forall, \exists\}^k\text{-}\text{TL}$ formula $\theta := Q_1 x_1 \dots Q_k x_k \cdot \varphi$ in a system \mathbf{P}^G is a Boolean combination of the truth values of the (non-indexed) TL formula φ , resulting from different valuations of the variables x_1, \dots, x_k . By the COMPOSITION property, two different topologies G, H , with corresponding valuations \bar{g}, \bar{h} , that yield TL-equivalent valuation TSs will admit the same truth values of φ in $\mathbf{P}^G, \mathbf{P}^H$. By the FINITENESS property, all the valuation TSs fall into finitely many TL-equivalence classes. Hence, given G , evaluating θ in \mathbf{P}^G amounts to evaluating a Boolean function (that depends only on G, Q_1, \dots, Q_k) over finitely many variables (one variable for each representative valuation TS); and evaluating θ with respect to \mathcal{G} amounts to evaluating a set of such functions (all using the same variables). Since there are only finitely many Boolean functions over a finite set of variables we obtain a cutoff.⁵ \square

3.1 The Composition Theorem

Theorem 3 (Composition). For all $d, k \in \mathbb{N}$, topologies G, H , processes $\mathbf{X}, \mathbf{Y} \in \mathcal{P}^{\text{FD}}$, $\bar{g} \in V_G^k$ and $\bar{h} \in V_H^k$: if $G[\bar{g}] \equiv_{\text{CTL}_d^* \setminus X} H[\bar{h}]$ and the initial states of \mathbf{X} and \mathbf{Y} are bisimilar, then $\mathbf{X}^G[\bar{g}] \equiv_{\text{CTL}_d^* \setminus X} \mathbf{Y}^H[\bar{h}]$.

⁴ Indeed, there are infinitely many $\text{CTL}_1^* \setminus X$ formulas that are pairwise logically-inequivalent. E.g., every finite word over $\{0, 1\}$ can be represented as an LTS, which itself can be axiomatised by a $\text{CTL}_1^* \setminus X$ formula that uses the U operator.

⁵ The existence of a cutoff is independent of whether \mathcal{G} is computable. However, deciding whether a given number is a cutoff may not be easy. Consider for example the limited setting of [2]: there exists a computable \mathcal{G} and a fixed \mathbf{P} such that it is impossible, given $k, d \in \mathbb{N}$ (even fixing $d = 1$), to compute a cutoff [2]. Nonetheless, by [3], in the same setting (and we believe that also in our broader setting) one can compute a cutoff for many natural parameterized topologies \mathcal{G} .

Proof (sketch). The proof has the following outline. Let s_0 and s'_0 be the initial states of $\mathbf{X}^G|\bar{g}$ and $\mathbf{Y}^H|\bar{h}$, respectively. By Proposition 1, it is enough to show that if the assumption of the theorem holds then $s_0 \equiv_d s'_0$. This is done by induction on d . For stating the inductive hypothesis we first need the following definition: given a system \mathbf{P}^G , a function $\beta : [T] \rightarrow \text{tokens}(s)$ that maps token numbers to vertices in G is a *token assignment at s* if, for every $v \in \text{tokens}(s)$, the number of tokens mapped to v is equal to the number of tokens at v according to s , i.e., $(\text{tok}^{|\beta^{-1}(v)|}, v) \in \Lambda(s)$, where Λ is the labelling of \mathbf{P}^G .

The d th Inductive Hypothesis. *For every global state s of $\mathbf{X}^G|\bar{g}$ and global state s' of $\mathbf{Y}^H|\bar{h}$, conclude that $s \equiv_d s'$ if the following two conditions hold:*

1. $s(g_i) \sim s'(h_i)$ for all $i \in [k]$, and
2. there exists a token assignment β at s , and there exists a token assignment β' at s' , such that for all $i \in [T]$ we have that $\beta(i) \equiv_d \beta'(i)$.

The first condition says that s and s' assign bisimilar states to matching processes in \bar{g} and \bar{h} ; the second condition says that s and s' have their tokens in nodes of G, H (respectively) that are equivalent according to $\equiv_d^{G[\bar{g}], H[\bar{h}]}$. The theorem follows by showing that s_0, s'_0 satisfy these assumptions.

For the induction base, observe that (by the first assumption in the inductive hypothesis) s and s' assign bisimilar local states to matching processes in \bar{g}, \bar{h} and thus, s, s' have the same labelling and are indistinguishable by a $\text{CTL}_0^* \setminus \mathbf{X}$ formula; now apply Proposition 1.

The main work in proving the inductive step is to satisfy the second condition in the definition of \equiv_d (Definition 1). This requires that, for every path π in $\mathbf{X}^G|\bar{g}$ starting in s , one can find a $(d - 1)$ -matching path π' in $\mathbf{Y}^H|\bar{h}$ starting at s' (and vice versa). Note that since our setup is symmetric we can ignore the “vice-versa” and only find π' given π . We construct π' using the general scheme graphically depicted in Figure 2.

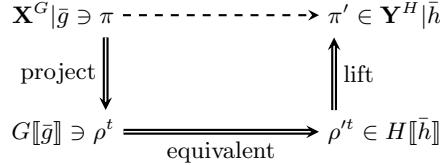


Fig. 2: Proving the COMPOSITION property via Definition 1.

First, given π , we assign to each token a unique number from 1 through T . This allows us to track the movements of individual tokens in G (according to the token-passing transitions of π) which we arbitrarily assume obey the following rule: all processes start in the initial vertex, and during a global token-passing transition, the smallest numbered token that the sending process has is the one being sent. Using this rule, for $i \in \mathbb{N}$ and $t \in [T]$ we can define the function $\text{token}_i : [T] \rightarrow V_G$ such that $\text{token}_i(t)$ is the vertex in which the token numbered t is located in the global state π_i . However, in order to construct π' so that it mimics π , we also need to know the directions the tokens take when entering and leaving nodes (which is information that is not explicitly present in π). The

reason this is needed is that processes in $\mathbf{X}^G|\bar{g}$ can change state based on the direction a token is sent to or received from (this is possible even if the process is direction-fair). To solve this, we arbitrarily choose some edge-path ξ in $\mathbf{X}^G|\bar{g}$ that induces π — being an edge-path, ξ contains the directions as part of each edge. As it turns out, we only need to know the directions of token-passing transitions affecting processes in \bar{g} . These are the send (resp. receive) directions of edges in G that start (resp. end) in a state in \bar{g} , and are captured by the extra nodes added to G to construct $G[\bar{g}]$. Thus, for each $t \in [T]$ we obtain from ξ a path ρ^t in $G[\bar{g}]$ that records the movement of token t along π .

For $t \in [T]$, by the assumption in the inductive hypothesis, $\beta(t) \equiv_d \beta'(t)$. Apply Definition 1 to the path ρ^t (that starts in $\beta(t)$) of $G[\bar{g}]$ to get a $(d-1)$ -matching path ρ'^t that starts in $\beta'(t)$ of $H[\bar{h}]$. The paths ρ'^1, \dots, ρ'^T will serve as the paths of tokens' movements for the path π' .

We construct π' by mimicking the transitions of π : an internal transition in $\mathbf{Y}^H|\bar{h}$ that involves a process $g_i \in \bar{g}$ is mimicked by a bisimilar internal transition using h_i ; and a token passing transition⁶ of the token number t (for $t \in [T]$) is mimicked as follows: we take the portion w of ρ^t this transition corresponds to, match it to a portion w' of ρ'^t (using the partitioning of ρ^t and ρ'^t into matching blocks), and then push the token t along an edge-path in H that induces w' .

The following lemma says that this last “pushing” step is possible.

Lemma.⁷ *Let $\mathbf{P} \in \mathcal{P}^{\text{FD}}$, let p, q be states of \mathbf{P} , and let G be a topology. Let $\rho = (v_1, (d_1, e_1), v_2)(v_2, (d_2, e_2), v_3) \dots (v_{m-1}, (d_{m-1}, e_{m-1}), v_m)$ be an edge-path that is a simple path (or a simple cycle) in G , and let s be a state of \mathbf{P}^G with $v_1 \in \text{tokens}(s)$. There exists a finite edge-path $\alpha = (f_0, \sigma_0, f_1)(f_1, \sigma_1, f_2) \dots (f_{h-1}, \sigma_{h-1}, f_h)$ in \mathbf{P}^G , with $f_0 = s$, such that:*

1. *α has $m-1$ token-passing transitions and in the i th token-passing transition v_i sends a token in direction d_i to v_{i+1} from direction e_i (for $0 \leq i < m$);*
2. *If vertex $x \in V_G$ is not on the path ρ , then no transition of α involves x .*

This lemma makes crucial use of the fact that \mathbf{P} is fair and direction-fair. Fairness ensures that tokens can always be made to flow in and out of a process, and direction-fairness ensures that tokens can always flow in any given direction. Indeed, the lemma is not true without both assumptions which is the main reason that without them the composition theorem does not hold and, as we show in Section 4, the PMC problem becomes undecidable. \square

3.2 FINITENESS property for $\text{CTL}_d^* \setminus \mathbf{X}$

Our aim in this section is to prove the FINITENESS property for $\text{CTL}_d^* \setminus \mathbf{X}$. We begin by recursively defining, given positive integers k, d and a k -valuation TS $G[\bar{g}] = \langle \text{AP}, Q, \{\text{init}\}, \delta, \lambda \rangle$, a marking function Ξ_d^k . This function associates with each vertex $v \in Q$ a $k+1$ -dimensional vector $\Xi_d^k(v)$ whose i th coordinate $\Xi_d^k(v)[i]$ is a set of strings over the alphabet $\cup_{u \in Q} \{\Xi_{d-1}^k(u)\}$. The marking function Ξ_d^k

⁶ Fortunately, we only have to mimic such transitions that cross blocks in ρ^t .

⁷ The full version of this lemma contains two more conclusions.

will help us later in defining the $\text{CTL}_d^*\backslash X$ -character of a valuation TS, which succinctly captures the $\text{CTL}_d^*\backslash X$ -equivalence class of this valuation TS.

Notation. The *positions* of a string v is the set $\{1, \dots, |v|\}$ if v is finite, and \mathbb{N} otherwise. A string w is *ultimately constant* if $|w| = \infty$ and $w_i = w_j$ for all $j \geq i$, for some i . Recall that the destuttering of a string is formed by removing identical consecutive letters. Define a mapping $\text{pos}_v : [|v|] \rightarrow [|v|]$ as follows: $\text{pos}_v(1) = 1$ and; for $i > 1$, $\text{pos}_v(i) := \text{pos}_v(i-1)$ if $v_{i-1} = v_i$, and otherwise $\text{pos}_v(i) := \text{pos}_v(i-1) + 1$. Intuitively, pos_v maps a position i of the string v to its corresponding position in $\text{destut}(v)$. Note that the image of pos_v is of the form $[L]$ for some $L \leq |v|$. Formally, $\text{destut}(v)$ is the string w of length L such that for all $i \leq L$, $w_i = v_{\min\{j: \text{pos}_v(j)=i\}}$. Thus, $v_i = w_{\text{pos}_v(i)}$ for all $i \leq L$. **The Marking** Ξ_d^k . Fix $k, d \in \mathbb{N}$, topology G , and k -tuple \bar{g} over V_G . Let $G[\bar{g}]$ be $\langle \text{AP}, Q, Q_0, \delta, \lambda \rangle$. For every vertex $v \in Q$, let v^\sim be the set of maximal paths in G starting in v that have no intermediate nodes in \bar{g} . Formally, a (finite or infinite) path $\pi = \pi_1 \pi_2 \dots$ is in v^\sim iff: $\pi_1 = v$ and, for all $1 < i < |\pi|$ we have $\pi_i \notin \bar{g}$ and, if π is finite then $|\pi| \geq 2$ and $\pi_{|\pi|} \in \bar{g}$. We write $v^{\sim 0} := \{\pi \in v^\sim \mid |\pi| = \infty\}$ for the infinite paths in v^\sim ; also, for every $i \in [k]$, we write $v^{\sim i} := \{\pi \in v^\sim \mid \pi_{|\pi|} = g_i\}$ for the set of paths in v^\sim that end in g_i .

In the definition below, for a (finite or infinite) path π , we write $\Xi_{d-1}^k(\pi) := \Xi_{d-1}^k(\pi_1) \Xi_{d-1}^k(\pi_2) \dots$ for the concatenation of the $d-1$ markings of the nodes of π . We define the marking Ξ_d^k of a node inductively (on d) as follows: $\Xi_0^k(v) := \lambda(v)$ and, for $d > 0$, $\Xi_d^k(v)$ is the vector $(\Xi_d^k(v)[0], \dots, \Xi_d^k(v)[k])$, where $\Xi_d^k(v)[i] := \cup_{\pi \in v^{\sim 0}} \{\text{destut}(\Xi_{d-1}^k(\pi))\}$ if $i = 0$; and $\Xi_d^k(v)[i] := \cup_{\pi \in v^{\sim i}} \{\text{destut}(\Xi_{d-1}^k(\pi')) \mid \pi' = \pi_1 \dots \pi_{|\pi|-1}\}$, for $1 \leq i \leq k$. That is, for $d = 0$, the marking $\Xi_d^k(v)$ is the label $\lambda(v)$; and for $d > 1$ the marking $\Xi_d^k(v)$ is a vector of sets of strings, where the i th coordinate of the vector contains the set of strings obtained by de-stuttering the Ξ_{d-1}^k markings of the nodes of paths in v^\sim (excluding the last node if $i > 0$) that end in g_i (if $i > 0$), or never visit any node in \bar{g} (if $i = 0$). Observe that, for every $0 \leq i \leq k$ and every $d > 0$, the marking $\Xi_d^k(v)[i]$ is a set of strings over the alphabet⁸ $\cup_{u \in Q} \{\Xi_{d-1}^k(u)\}$, and that all strings in $\Xi_d^k(v)[i]$ start with the letter $\Xi_{d-1}^k(v)$.

Since, for all $0 \leq i \leq k$ and $d > 0$, all strings in $\Xi_d^k(v)[i]$ start with the letter $\Xi_{d-1}^k(v)$, and since $\Xi_d^k(v)[i] = \emptyset$ iff $v^{\sim i} = \emptyset$, we get the following lemma:

Lemma 1. *For every $d > 0$, if v, u are nodes (of possibly different k -valuation TSs) such that $\Xi_d^k(v) = \Xi_d^k(u)$, then for all $0 < j \leq d$ we have that $\Xi_j^k(v) = \Xi_j^k(u)$. If, in addition, $v^\sim \neq \emptyset$ then also $\Xi_0^k(v) = \Xi_0^k(u)$.*

The $\text{CTL}_d^*\backslash X$ -character of a valuation TS. Given a k -valuation TS $G[\bar{g}] = \langle \text{AP}, Q, \{\text{init}\}, \delta, \lambda \rangle$, the $\text{CTL}_d^*\backslash X$ -character of $G[\bar{g}]$ is defined as the following vector $(\langle \lambda(\text{init}), \Xi_d^k(\text{init}) \rangle, \langle \lambda(g_1), \Xi_d^k(g_1) \rangle, \dots, \langle \lambda(g_k), \Xi_d^k(g_k) \rangle)$ of the pairs of labels and Ξ_d^k markings of the initial state and the states in \bar{g} .

The following theorem relates the $\text{CTL}_d^*\backslash X$ -character of a valuation TS and its $\text{CTL}_d^*\backslash X$ -equivalence class.

⁸ Here, the empty set \emptyset is a letter in $2^{[k]}$, not to be confused with the empty string ϵ .

Theorem 4. For every $k, d \in \mathbb{N}$, if $G[\bar{g}], H[\bar{h}]$ are two k -valuation TSs with the same $\text{CTL}_d^* \setminus X$ -character, then $G[\bar{g}] \equiv_{\text{CTL}_d^* \setminus X} H[\bar{h}]$.

Our next goal is to prove that there are (for given $k, d \in \mathbb{N}$) only finitely many $\text{CTL}_d^* \setminus X$ -characters for all k -valuation TSs. We do this by showing that (for fixed alphabets $\Sigma_{\text{snd}}, \Sigma_{\text{rcv}}$) for all k, d , all k -valuation TSs $G[\bar{g}]$, and all $v \in G$, we have that $\Xi_d^k(v)$ ranges over finitely many values. This is clearly true for $d = 0$. For $d > 0$, we prove this by defining a finite poset Υ_d^k , which depends only on k and d , and showing that $\Xi_d^k(v) \in \Upsilon_d^k$. We begin by defining a relation \preceq between sets of strings.

Definition of \preceq . For sets of strings $X, Y \subseteq (\Sigma^+ \cup \Sigma^\omega)$, define $X \preceq Y$ if for all $x \in X$ there exists $y \in Y$ such that x is a (not necessarily proper) suffix of y .

It is easy to verify that the relation \preceq is reflexive and transitive, but that it may not be antisymmetric (consider for example $X = \{b, ab\}$ and $Y = \{ab\}$).

Lemma 2. Given a k -valuation TS $G[\bar{g}]$, and a path $\pi_1 \dots \pi_t$ in it satisfying $\pi_l \notin \bar{g}$ for all $1 < l \leq t$, we have that: $\Xi_d^k(\pi_j)[i] \preceq \Xi_d^k(\pi_h)[i]$ for every $0 \leq i \leq k$, and $d > 0$, and $1 \leq h < j \leq t$.

The relation \preceq is antisymmetric when restricted to the domain consisting of sets of strings Z such that: (i) all strings in Z start with the same letter $\text{first}(Z)$ (i.e., there exists $\text{first}(Z) \in \Sigma$ such that for all $w \in Z$, $w_1 = \text{first}(Z)$); (ii) in every string in Z the letter $\text{first}(Z)$ appears only once (i.e., for all $w \in Z$, $i > 1$ implies $w_i \neq \text{first}(Z)$). Given an alphabet Σ , let $\mathbb{P}_\Sigma \subset 2^{\Sigma^+ \cup \Sigma^\omega}$ denote the set of all sets of strings Z (over Σ) satisfying the above two conditions. We have:

Lemma 3. $(\mathbb{P}_\Sigma, \preceq)$ is a partially ordered set.

Definition of $(\Upsilon_d^k, \preceq_d)$. The definition is by induction on d : for $d = 0$ we have $\Upsilon_0^k := 2^{\text{AP}}$ (recall that $\text{AP} = [k] \cup \Sigma_{\text{snd}} \cup \Sigma_{\text{rcv}}$); and \preceq_0 is the transitive closure of the relation obtained by having, for every $X \in 2^{[k]}$, every $\mathbf{d} \in \Sigma_{\text{snd}}$, and every $\mathbf{e} \in \Sigma_{\text{rcv}}$, that: $\{\mathbf{d}\} \preceq_0 X, \{\mathbf{d}, \mathbf{e}\} \preceq_0 X, \emptyset \preceq_0 \{\mathbf{d}\}$, and $\{\mathbf{e}\} \preceq_0 \emptyset$. For $d > 0$, let: $\Upsilon_d^k = \{X \in (\mathbb{P}_{\Upsilon_{d-1}^k})^{k+1} \mid w \in X[i] \text{ implies } w_{j+1} \prec_{d-1} w_j \text{ for all } 0 \leq i \leq k \text{ and } 1 \leq j < |w|\}$ and take \preceq_d to be the point-wise ordering of vectors, i.e., $X \prec_d Y$ iff $X[i] \preceq Y[i]$ for every $0 \leq i \leq k$, where \preceq is the ordering defined earlier for sets of strings. Intuitively, $X \in \Upsilon_d^k$ iff every coordinate of X contains strings over the alphabet Υ_{d-1}^k that all start with the same letter and are all strictly decreasing chains of the poset $(\Upsilon_{d-1}^k, \preceq_{d-1})$. Observe that if Υ_{d-1}^k is a finite set then there are finitely many strictly decreasing chains (each of finite length) in $(\Upsilon_{d-1}^k, \preceq_{d-1})$, implying that Υ_d^k is also finite. Since Υ_0^k is finite, we can conclude, for every $d \geq 0$, that Υ_d^k is a finite set of finite strings.

The following lemma states that for fixed k, d (recall that we assume fixed alphabets $\Sigma_{\text{snd}}, \Sigma_{\text{rcv}}$) the domain of Ξ_d^k is contained in Υ_d^k (and is thus finite). Note that this also implies that even though the strings in $\Xi_d^k(v)[0]$ are obtained by de-stuttering markings of infinite paths in $v^{\rightsquigarrow 0}$ they are all finite strings.

Lemma 4. For all k, d , if v is a vertex of a k -valuation TS then $\Xi_d^k(v) \in \Upsilon_d^k$.

We conclude with the finiteness theorem for $\text{CTL}_d^* \setminus X$.

Theorem 5 (Finiteness). *For every $k, d \in \mathbb{N}$, the set $\{G[\bar{g}] : G \text{ is a topology, } \bar{g} \in V_G^k\}$ has only finitely many $\equiv_{\text{CTL}_d^* \setminus X}$ equivalence classes.*

Proof. The theorem follows immediately from the fact that the $\text{CTL}_d^* \setminus X$ -character of a valuation TS is a finite vector, Lemma 4, and Theorem 4.

4 Undecidability

The positive decidability results appearing in Section 3 are the strongest one can hope for. Indeed, we prove that if one drops any of the restrictions that were imposed on the process template, namely of fairness and direction/value-fairness, then PMC becomes undecidable. Furthermore, these undecidability results hold even if multiple other strong restrictions are put instead (such as having a single token, having no values, having one send or one receive direction, etc.)

Our proofs reduce the non-halting problem for counter-machines (CMs) to the PMC problem. The basic encoding uses one process (the *controller*) to orchestrate the simulation and store the line number of the CM, and many *memory* processes, each having one bit for each counter. The main difficulty we face, compared to other reductions that follow this basic encoding (e.g., in [32,18,20,2]), is how to make sure that the controller’s commands are executed by the memory processes given that the restrictions imposed in the theorems prevent the controller from communicating its commands to the memory processes.

Theorem 6. *Let \mathcal{P}^{DV} denote the set of process templates that are direction/value-fair but not necessarily fair. There exists \mathcal{G} such that $\text{PMCP}_{\mathcal{G}}(\mathcal{P}^{\text{DV}}, \{\forall\}^5\text{-LTL} \setminus X)$ is undecidable, even if one limits the processes to have a single valueless token (i.e. $T = 1$ and $|\Sigma_{\text{val}}| = 1$), and with a single receive direction (i.e., $|\Sigma_{\text{recv}}| = 1$). The same holds replacing “receive” by “send”; furthermore, \mathcal{G} is computable.*

A template \mathbf{P} is *receive-direction fair* if for every i -sending state q and for every $d \in \Sigma_{\text{recv}}$, there is a finite i -path from q ending in a state that is ready to receive in direction d ; it is *send-direction fair* if the previous condition holds with “send(ing)” replacing “receive(ing)” and Σ_{snd} replacing Σ_{recv} ; it is *direction-fair* if it is both receive- and send-direction fair. A template \mathbf{P} is *value-fair* if for every i -receiving (resp. i -sending) state q , and for every token-value $m \in \Sigma_{\text{val}}$, there is a finite i -path from q ending in a state that is ready to receive (resp. send) value m . It is important to note that a template that is both direction-fair and value-fair is *not*, in general, direction/value-fair. The difference is that while the former can correlate values with directions, the latter cannot. For example, it may be that from every state it can only receive/send in direction a if the value of the token is 0, and receive/send in direction b only if the token value is 1. This kind of behaviour is not allowed if the template is direction/value-fair.

Theorem 7. *Let \mathcal{P}^F be the set of process templates that are fair but not necessarily direction/value-fair. There exists \mathcal{G} such that $\text{PMCP}_{\mathcal{G}}(\mathcal{P}^F, \{\forall\}^5\text{-LTL} \setminus X)$ is undecidable, even for direction fair and value fair templates with $|\Sigma_{\text{recv}}| = 1$; furthermore, \mathcal{G} is computable.*

5 Discussion

The literature contains PMC decidability results of token-passing systems with a single token [18,13,16,2,3], and with multiple tokens [16,22].⁹ However, the results on multiple tokens (and their proofs) only apply to linear-time specifications, and only to ring or clique network-graphs. In contrast, our results apply to branching-time specifications and to general network-graphs.

The proof of our decidability result follows the framework outlined in [2] (inspired by [13,18]) which suggests combining composition and finiteness results. Moreover, our work inherits from [13,2] the non-uniformity of the decision problem. We leave for future work the problem of calculating explicit cutoffs for concrete classes of network-graphs, as was done in [3].

Rabinovich [28] also uses the composition method for solving PMC. He considers the PMC problem for propositional modal logic assuming the parameterized network-graphs \mathcal{G} have a decidable monadic-second order validity problem. While the systems in [28] are very general, the specification language, i.e., modal logic, is orthogonal to ours (e.g., it can not express liveness properties).

To the best of our knowledge, [28,2] are the only other works that use composition to establish decidability of PMC of distributed systems. While proving composition and finiteness may not be easy, we find the methodology to be elegant and powerful. Indeed, in all of these cases, no other method is known (e.g., automata, tableaux) for proving decidability. We leave for future work the intriguing problem of applying this methodology to other problems.

References

1. P.A. Abdulla, G. Delzanno, O. Rezine, A. Sangnier, and R. Traverso. On the verification of timed ad hoc networks. In *FORMATS*, pages 256–270, 2011.
2. B. Aminof, S. Jacobs, A. Khalimov, and S. Rubin. Parameterized model checking of token-passing systems. In *VMCAI*, pages 262–281, 2014.
3. B. Aminof, T. Kotek, F. Spegni, S. Rubin, and H. Veith. Parameterized model checking of rendezvous systems. In *CONCUR*, pages 109–124, 2014.
4. B. Aminof, A. Murano, S. Rubin, and F. Zuleger. Verification of asynchronous mobile-robots in partially-known environments. In *PRIMA 2015*, pages 185–200, 2015.
5. B. Aminof, A. Murano, S. Rubin, and Florian Zuleger. Automatic verification of multi-agent systems in parameterised grid-environments. In *AAMAS*, 2016.
6. B. Aminof, S. Rubin, F. Spegni, and F. Zuleger. Liveness of parameterized timed networks. In *ICALP*, pages 375–387, 2015.
7. B. Aminof, S. Rubin, and F. Zuleger. On the expressive power of communication primitives in parameterised systems. In *LPAR*, pages 313–328, 2015.
8. K. Apt and D. Kozen. Limits for automatic verification of finite-state concurrent systems. *Information Processing Letters*, pages 307–309, 1986.
9. C. Baier and J-P. Katoen. *Principles of model checking*. MIT Press, 2008.

⁹ Communication in [22] is by rendezvous, powerful enough to express token-passing.

10. R. Bloem, S. Jacobs, A. Khalimov, I. Konnov, S. Rubin, H. Veith, and J. Widder. *Decidability of Parameterized Verification*. Synthesis Lectures on Distributed Computing Theory. M&C, 2015.
11. M. C. Browne, E. M. Clarke, and O. Grumberg. Reasoning about networks with many identical finite state processes. *Inf. Comput.*, pages 13–31, April 1989.
12. K. M. Chandy and J. Misra. The drinking philosophers problem. *ACM TOPLAS*, 6(4):632–646, 1984.
13. E. Clarke, M. Talupur, T. Touili, and H. Veith. Verification by network decomposition. In *CONCUR 2004*, pages 276–291, 2004.
14. S. Das. Mobile agents in distributed computing: Network exploration. *Bull. EATCS*, pages 54–69, 2013.
15. S. Demri and D. Poitrenaud. Verification of infinite-state systems. In S. Haddad, F. Kordon, L. Pautet, and L. Petrucci, editors, *Models and Analysis in Distributed Systems*, chapter 8, pages 221–269. Wiley, 2011.
16. E. A. Emerson and Vineet Kahlon. Parameterized model checking of ring-based message passing systems. In *CSL*, pages 325–339. Springer, 2004.
17. E.A. Emerson and V. Kahlon. Model checking guarded protocols. In *LICS*, pages 361–370. IEEE, 2003.
18. E.A. Emerson and K.S. Namjoshi. Reasoning about rings. In *POPL*, pages 85–94, 1995. Journal version: Int. J. Found. Comp. Sci., 14 (4), 2003.
19. E.A. Emerson and A. Sistla. Symmetry and model checking. In *CAV*, pages 463–478, 1993.
20. J. Esparza, A. Finkel, and R. Mayr. On the verification of broadcast protocols. In *LICS*, pages 352–359. IEEE, 1999.
21. S. Feferman and R.L. Vaught. The first-order properties of algebraic systems. *Fund. Math.*, 47:57–103, 1959.
22. S. German and A. Sistla. Reasoning about systems with many processes. *JACM*, 39(3):675–735, 1992.
23. S. Ghilardi, E. Nicolini, S. Ranise, and D. Zucchelli. Combination methods for satisfiability and model-checking of infinite-state systems. In F. Pfenning, editor, *Automated Deduction – CADE-21*, pages 362–378, 2007.
24. T. Herman. Probabilistic self-stabilization. *Inf. Process. Lett.*, 35(2):63–67, 1990.
25. A. John, I. Konnov, U. Schmid, H. Veith, and J. Widder. Parameterized model checking of fault-tolerant distributed algorithms by abstraction. In *FMCAD*, pages 201–209, 2013.
26. A. Kosowski. *Time and Space-Efficient Algorithms for Mobile Agents in an Anonymous Network*. Habilitation, U. Sciences et Technologies - Bordeaux I, 2013.
27. E. Kranakis, D. Krizanc, and S. Rajsbaum. Computing with mobile agents in distributed networks. In S. Rajasekaran and J. Reif, editors, *Handbook of Parallel Computing: Models, Algorithms, and Applications*. CRC Press, 2007.
28. A. Rabinovich. On compositionality and its limitations. *ACM TOCL*, 8(1), 2007.
29. S. Rubin. Parameterised verification of autonomous mobile-agents in static but unknown environments. In *AAMAS*, pages 199–208, 2015.
30. S. Shamir, O. Kupferman, and E. Shamir. Branching-depth hierarchies. *ENTCS*, 39(1):65 – 78, 2003.
31. S. Shelah. The monadic theory of order. *Ann. of Math.*, pages 379–419, 1975.
32. I. Suzuki. Proving properties of a ring of finite-state machines. *Inf. Process. Lett.*, 28(4):213–214, 1988.

Decidability in Parameterized Verification*

Roderick Bloem¹ Swen Jacobs² Ayrat Khalimov¹ Igor Konnov³
Sasha Rubin⁴ Helmut Veith³ Josef Widder³

¹ TU Graz

² Universität des Saarlandes

³ TU Wien

⁴ Università degli Studi di Napoli “Federico II”

Abstract

Parameterized model checking is an active research field that considers automated verification of distributed or concurrent systems, for all numbers of participating processes. In our recent book [11] we surveyed literature on decidability of parameterized verification. The system models, as well as the proof methods, that are studied in this field are similar to those from distributed computing. For instance, we survey results for token passing systems and ad hoc networks. The proof methods for undecidability include simulation of two-counter machines. For decidability, we survey cut-off results that provide conditions to reduce parameterized verification to model checking of fixed-size systems. We believe that the results are of interest to the readers of the Distributed Computing Column, and in this short note we want to give a taste of parameterized model checking.

1 Introduction

Designing concurrent or distributed systems and proving their correctness is both difficult and error-prone. There are, roughly speaking, two streams of research that address these tasks. On the one hand, principles of distributed computing are studied in order to develop mathematical proof methods can be used establish complexity results in distributed computing (lower and upper bounds) [34, 8], and to prove the correctness of distributed algorithms. On the other hand, one investigates computer aided verification methods, such as model checking [14]. Mathematical proofs typically consider parameterized systems, where, e.g., the parameter n determines the number of replicas in a distributed system, and one determines the correctness of a distributed algorithm for all values of n . Historically, model checking considers fixed size and finite state systems only. For instance, one fixes $n = 4$ and checks the small system for presence of bugs. However, nowadays most research in model checking considers parameterization in one way or another, e.g., parameterized number of processes, or parameterized domain of variables. Consequently, there is a vast literature in this domain.

*We dedicate this article to the memory of Helmut Veith, who passed away tragically while this manuscript was being prepared. His curiosity and energy ignited our joint long-term effort in parameterized verification.

This work was supported by the Austrian National Research Network RiSE (S11403, S11405, S11406) and project PRAVDA (P27722) of the Austrian Science Fund (FWF), by the Vienna Science and Technology Fund (WWTF) through grants APALACHE (ICT15-103) and PROSEED, by the German Research Foundation (DFG) through SFB/TR 14 AVACS and project ASDPS (JA 2357/2-1), and by the Istituto Nazionale di Alta Matematica through INdAM-COFUND-2012, FP7-PEOPLE-2012-COFUND (Proj. ID 600198).

Our recent book [11] focuses on automatic verification of systems where the number of processes is parameterized, while the local state space of the processes is finite. More precisely, we consider the decidability of parameterized model checking of concurrent systems. The literature in this field considers different computational models and we thus survey results for the most studied models: token passing systems, systems where processes coordinate by pairwise rendezvous or one-to-many synchronization (broadcast), guarded protocols, and finally ad-hoc networks.

In this note, we briefly discuss the features of these computational models and some conclusions we have drawn from surveying the literature. For formal definitions as well as a detailed survey of the results and their proofs we refer to the book. We start with discussing the computational model used in the book in Section 2 and some often-used proof techniques in Section 3, and then describe the different system models that are covered in the book, and summarize their decidability results in Sections 4 to 7.

2 Models and Specifications

Finding out whether parameterized model checking is decidable for one’s favorite computational model boils down to checking whether it is captured by the semantics of one of the published computational models. However, computational models are scattered over the literature and use different terminology, and results are based on slightly different assumptions. Hence, it is cumbersome to check whether existing results apply to our favorite models. Our motivation was to provide a one-stop-source for results that eases this task. This required us to provide a definition for concurrent systems that incorporates many of the foundational computational models in the parameterized model checking literature. The models we discuss in Sections 4 to 7 specialize specific features of this general model.

In our framework, a concurrent system, or *system instance*, \overline{P}^G is composed of a vector of *process templates* $\overline{P} = (P^1, \dots, P^d)$, and a graph G , on which copies of the templates are arranged. We consider discrete time, and at each time step either one process acts alone, or some process v initiates an action and some set of processes that are connected to v in G simultaneously synchronize with v . Thus, we consider that processes coordinate instantaneously using synchronization primitives (e.g., rendezvous), and do not consider, e.g., non-blocking communication by message buffers. Most importantly, the process templates we consider have a fixed and finite local state space.

Synchronization primitives can then be defined by restricting the number of processes that can simultaneously synchronize with the initiating process. We do so by a so-called *synchronization constraint* that specifies how many processes should synchronize. For instance, pairwise rendezvous is captured by having the singleton $\{1\}$ as synchronization constraint, which means that every synchronous transition must be taken by the initiating process and exactly one other process.

Then, a *parameterized system* is a sequence of system instances formed from a fixed vector of process templates and a sequence of graphs \mathbf{G} . Typical sequences of graphs are rings of size n , cliques of size n , or stars of size n , where n is the parameter. The n th instance of a parameterized system is the system instance $\overline{P}^{G(n)}$. *Parameterized specifications* make statements about parameterized systems by quantifying over process indices (i.e., vertices of $\mathbf{G}(n)$), e.g., “every process v of $\mathbf{G}(n)$ eventually satisfies predicate p ”. To formalize this, we recall the definitions of several *indexed temporal logics* in the book. The *parameterized model checking problem (PMCP)* is to decide whether for all $n \in \mathbb{N}$ the instance $\overline{P}^{G(n)}$ satisfies the specification.

3 Proof Techniques

3.1 Undecidability

In this area, undecidability is typically proven by reduction from the non-halting problem of two-counter machines, which is known to be undecidable. In the proof, one then shows how to simulate up to n steps of a two-counter machine in a system composed of $n + 1$ finite-state processes: There is a *controller* process that simulates the internal state of the two-counter machine. The remaining n processes collectively encode the counter values in a way that allows for storing values up to n , which is the maximal value that can be written in n steps. The proofs differ in how the controller uses the synchronization primitives to issue commands to increment/decrement counters and test counters for zero.

The prototypical such proof in a token ring is by Emerson and Namjoshi [27] and describes a distributed algorithm that performs this task. This improved the first undecidability results for systems consisting of identical processes arranged in a uni-directional ring with a single multi-valued token by Suzuki [41] who reduces non-halting of Turing machines to parameterized model checking. Undecidability of parameterized model checking then follows from undecidability of the non-halting problem.

3.2 Decidability

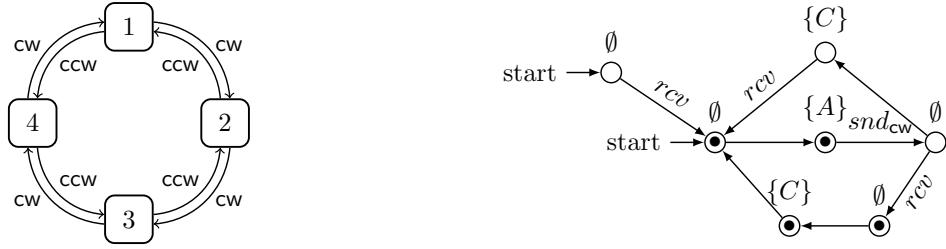
One way to prove decidability of parameterized verification for a certain class of concurrent systems is to show that this class can be represented as a well-structured transition system [29]. For such systems Abdulla et al. [2] provided a comprehensive theory.

A quite intuitive approach to parameterized model checking is to check the system for small values of n and *assume* that if there is a bug in a system with a large number of components, then the bug already appears in the small system. This approach can be formalized as a *cutoff* or *decomposition* statement [27, 15] that reduces the parameterized model checking problem to a finite collection of classic model checking problems. For specific classes of systems one can show that such cutoffs or decompositions indeed exist, from which decidability of parameterized model checking for such systems follows. To prove that c is a cutoff for a class of systems and specifications in an indexed temporal logic, it suffices to show that a system with exactly c components relates to a system with arbitrarily many components. The details of the specific relation (e.g., simulation or bisimulation) depend on the class of systems and the temporal logic under consideration.

4 Token-passing Systems

In a token-passing system (TPS), processes communicate by passing a token in the network. The token may or may not carry a value that can be updated by the process that holds it. In case of a valueless token, the only purpose of the token is to distinguish the process that currently holds it from all the other processes, which allows one for example to implement systems with mutual exclusion properties.

A well-known example system that can be modeled as a TPS with valueless token is *Milner's scheduler* [36]. The scheduler is composed of an arbitrary number of components, each of which is responsible for activating some (unspecified) task and receives confirmation when its task has been executed. Scheduling should ensure that the tasks are activated in a round-robin scheme, and that every task has terminated before being activated again. As noted by Emerson and Namjoshi [27], this can naturally be modeled in a token ring. That



(a) A bi-directional ring with 4 nodes. Edges are labeled with directions cw (clockwise) or ccw (counter-clockwise). Processes can send the token in a direction by using actions snd_{cw} or snd_{ccw} , respectively.

(b) Process template for Milner's scheduler. States with token are depicted with a dot inside. State labels are depicted above states. Label A stands for activation of the task, C for completion of the task.

Figure 1: Milner's scheduler in a token ring.

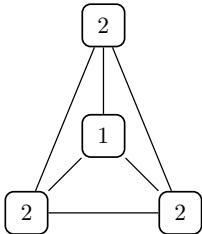
is, processes are arranged in a ring, and one process starts with the token. A process that has the token will activate its task, and then send the token to the next process. After starting the task, it will wait for both the return of the token and a confirmation that the task has terminated. Only after both of these events, the process will activate its task again. Thus, communication by a single valueless token is sufficient to guarantee the global properties required by the Milner scheduler. The graph and process template for Milner's scheduler in a token ring are depicted in Figure 1.

For other classes of systems, the parameterized verification literature usually considers a single fixed graph structure, like cliques for systems with broadcast communication or guarded protocols. In contrast, TPSs have been analyzed on a variety of graphs that can be much more complex. In particular, this includes graphs where connections may be labeled with directions, and processes can choose into which direction they want to send the token. Finally, another orthogonal extension of the model considers systems with multiple tokens.

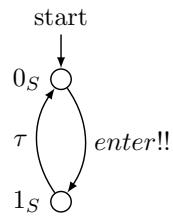
Results. Most of the decidability results in the literature are for TPSs with a single valueless token [27, 15, 5], and where token passing satisfies some notion of fairness. However, even in systems with such a restricted communication primitive, parameterized model checking is undecidable if we consider arbitrarily complex graphs and specifications in a branching-time logic. Undecidability proofs show that such systems can simulate two-counter machines, and deciding the PMCP would decide the non-halting problem of these machines [15, 5]. To obtain decidability results, the literature considers restrictions on the graph structure, as well as on the number of index and path quantifiers in the specification.

For rings, Emerson and Namjoshi [27] have identified concrete cutoffs between 2 and 5, depending on the specification, but independent of the process template. Cutoffs for rings and some other classes of graphs can be found by manually constructing a bi-directional simulation as mentioned in Section 3.2 [27, 5]. In TPSs with possibly complex graphs, this simulation needs to take into account whether processes mentioned in the specification are direct neighbors or not, and consider all possible cases of neighborship relations if the specification quantifies over multiple processes. For certain classes of graphs that can be constructively specified, cutoffs can also be computed automatically [6].

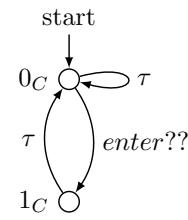
If tokens can carry values, decidability is lost even in uni-directional rings [41]. Similarly, if processes can distinguish directions in the graph, we get undecidability in bi-directional rings [5]. To recover decidability, additional restrictions on graphs, processes, or specifications are necessary.



(a) A clique graph with three clients and one server.



(b) A server process.



(c) A client process.

Figure 2: A Client/Server system.

5 Broadcast and Asynchronous/Pairwise-Rendezvous

The broadcast communication primitive is an abstraction of, e.g., ethernet-like broadcast, GSM’s cell-broadcast, communication in Prasad’s Calculus of Broadcasting Processes (CBP) [39], or the `notifyAll` method in Concurrent Java [18]. To use the broadcast communication primitive, an initiator process sends a message and every process that is able to receive the message (if any) immediately and simultaneously does so. Figure 2 illustrates a client-server system with three clients. If the server broadcasts the message *enter* by firing the transition from state 0_S to 1_S , then, simultaneously, every client that is in state 0_C transitions to 1_C .

Two other related primitives are asynchronous-rendezvous in which exactly one process that is able to receive the message does so (if any), and pairwise-rendezvous which is like asynchronous-rendezvous except that sending is blocked if there is no process that is able to receive. Asynchronous-rendezvous is an abstraction of, e.g., the `notify` method in Concurrent Java [18], and pairwise-rendezvous is like synchronized communication in Milner’s Calculus of Communicating Systems (CCS) and Hoare’s Communicating Sequential Processes (CSP).

Broadcast can express many others primitives including asynchronous- and pairwise-rendezvous, token-passing (Section 4) and disjunctive guards (Section 6) [7].

Results. We assume the communication graph is a clique (other graphs are assumed in Sections 4 and 7). Parameterized model checking of broadcast systems is undecidable for liveness properties (e.g., some state from a given set is seen infinitely often), and decidable for safety properties (e.g., no state from a given set is ever visited). The proof of undecidability uses the basic encoding of two counter machines outlined above. The interesting part of the simulation is testing a counter for zero, and uses a trick in which the controller broadcasts a guess of whether the counter is zero or not, thus potentially introducing errors to the simulation, which can be compensated for, however. Decidability of safety properties follows from the fact that broadcast systems are well-structured transition systems [2]. The case of asynchronous-rendezvous is similar.

On the other hand, for pairwise-rendezvous, both liveness and safety specifications are decidable. The proof of this fact shows that such systems can be expressed as Petri Nets or Vector Addition Systems [28]. The main idea is to use a counter-representation that captures, for each configuration, the number of processes in each state. Thus configurations are represented as vectors of natural numbers, and a rendezvous between two processes corresponds to adding a fixed vector to a configuration.

All the results above are for linear-temporal specifications. Branching-temporal specifications are undecidable already for pairwise rendezvous [6].

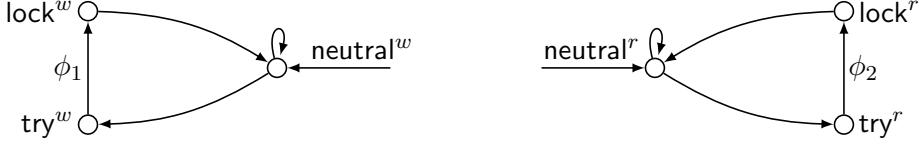


Figure 3: A readers/writer protocol.

6 Guarded Protocols

The key feature of guarded protocols is that every process can query the local states of the other processes in an anonymous way. For instance, a process j evaluates the disjunctive guard $[\exists \text{ other } i] A_i \vee B_i$ to true, whenever a global state contains a process i (other than j) that is either in local state A , or B . Likewise, a process j evaluates the conjunctive guard $[\forall \text{ other } i] A_i \wedge B_i$ to true, whenever all processes different from j are either in A , or in B . While the seminal paper by Emerson and Namjoshi [26] introduced guarded protocols for synchronous systems, the follow-up papers by Emerson and Kahlon [23, 24, 25] considered asynchronous guarded protocols.

Guarded protocols have been used to model cache-coherence protocols. A more classical example is a *multiple* readers and *single* writer protocol shown in Figure 3. The readers start in the state neutral^r , and the writers start in the state neutral^w . A writer's transition from the trying state try^w to the locking state lock^w is guarded with the guard $\phi_1 \equiv [\forall \text{ other } j] \text{neutral}_j^r \vee \text{neutral}_j^w \vee \text{try}_j^r \vee \text{try}_j^w$, which forbids this writer to make a transition, if some readers or writers are in the locking state. A reader makes a transition from the trying state try^r to the locking state lock^r , *only* if all writers are in the neutral state, that is, the guard ϕ_2 is $[\forall \text{ other } j] \text{neutral}_j^r \vee \text{neutral}_j^w \vee \text{try}_j^r \vee \text{lock}_j^r$.

Results. Unfortunately, parameterized model checking of boolean guarded protocols — that is, protocols that use both disjunctive and conjunctive guards — is undecidable [26]. As usual, undecidability is shown by simulating two-counter machines. Thus, the research has been focused on verification of systems that have only one kind of guards [23, 24, 25].

As shown by Emerson and Kahlon [23], each system composed of n processes having Q local states and using only disjunctive guards has a cutoff of size $|Q| + 2$. Hence, to check a temporal property for all system sizes, it is sufficient to check the property for systems of size up to $|Q| + 2$. A cutoff of size $2|Q| + 1$ was also found [23] for systems with conjunctive guards having the following restrictions: every process can remain indefinitely long in the initial state, and no guard evaluates to false due to a process staying in the initial state.

Further, Emerson and Kahlon showed that systems with disjunctive guards can be simulated by systems with rendezvous communication [25]. Hence, one achieves decidability of PMCP for the systems that use both disjunctive guards and rendezvous communication.

In most cases, extending guarded protocols with broadcast leads to undecidability [25]. The only known decidability result is for regular action-based properties [25], that is, the properties that can be described with a finite automaton recognizing words over the alphabet of action labels.

Finally, Emerson and Kahlon developed special techniques for the restricted versions of guarded protocols that model cache coherence protocols [24].

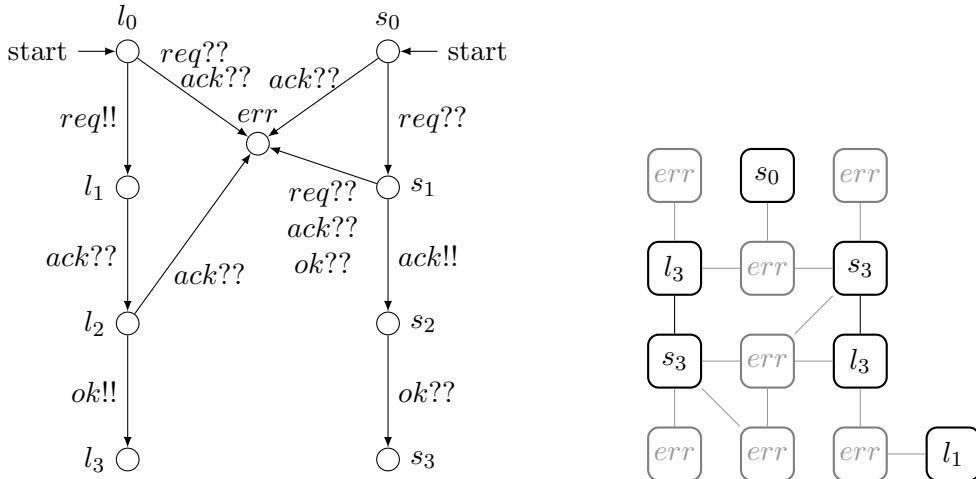


Figure 4: On the left: Request-Acknowledgment-OK protocol for ad hoc networks [19]: initial states are $\{s_0, l_0\}$, $a!!$ means “broadcast message a ”, and $a??$ means “receive message a ” (for $a \in \{req, ack, ok\}$). On the right: example state after running the protocol.

7 Ad Hoc Networks

In ad hoc networks, processes communicate using “local” broadcasts: such messages are visible only to processes in the communication range of the sender. Ad hoc networks can be captured by broadcast systems (Section 5) if we model the communication ranges using the system graph: two nodes are connected only if in the ad hoc network the two processes can hear each other.

Figure 4 illustrates a Request-Acknowledgment-OK protocol to be run by processes of an ad hoc network, and a possible state after executing the protocol. The intention of the protocol is to “create” pairs of connected processes in states l_3 and s_3 that are isolated from others by processes in state err .

In contrast to other systems considered in our survey, the literature on ad hoc networks [19, 20, 21, 1] does not study the PMCP for a general class of specifications, but rather three specific problems called COVER, REPEAT, and TARGET. COVER asks, given a process description and a process control state, whether the control state can be reached in a network of some size. Similarly, REPEAT asks if the state can be reached infinitely often, and TARGET asks if all processes can simultaneously reach the state. The first two problems can be expressed as a PMCP using indexed temporal logic with index quantifiers in prenex form, but the last one requires a non-prenex fragment.

The literature also differentiates the results for different classes of graphs. We survey results for five classes of undirected graphs, inspired by those found in practice, including cliques, wheels, stars, processes arranged in clusters, and unrestricted graphs. We also survey results [1] for directed graphs. Such graphs can model discrepancies in the send and receive transmission ranges of processes.

Finally, the literature distinguishes ad hoc networks with and without failures. The failure model is: any broadcast message can be lost non-deterministically for some of the recipients. A related model is that of mobile networks, in which the network graph can non-deterministically change during the execution. Delzanno et al.[21] showed that the models of lossy and mobile networks are equivalent, so we focus on the former only.

Results. Roughly, COVER for ad hoc networks without failures is undecidable on graphs that can have simple paths of unbounded length, except cliques (e.g., wheel-like graphs) [19]. The undecidability proof uses the ability of the network to simulate a given two-counter machine, as follows. First, the network runs a special protocol that shuts down all processes except one leader connected to two lists of processes. Then, the leader simulates the control of the two-counter machine using the two lists to store the values of the counters. In this construction, to be able to store the unbounded values of the counters, the two lists should be of the unbounded length (unbounded in the parameterized system).

COVER is decidable on graphs in which all simple paths have a bounded length [19, 20]. The decidability proofs go via the machinery of well-structured transitions systems. Intuitively, the proofs use the insight that behaviours of an infinite family of networks have a finite basis, and it can be found using a saturation algorithm. Note that REPEAT and TARGET are undecidable on all graph classes we survey, if considered on ad hoc networks without failures [19].

All three problems become decidable for the case of lossy ad hoc networks [21]. The hostile failure model makes it impossible to simulate a two-counter machine, and the decidability proofs go via reduction to Petri nets (a certain kind of well-structured transition systems).

8 Conclusions

We believe that parameterized model checking and distributed computing could both benefit from studying parameterized verification in the context of state-of-the-art computation models for distributed algorithms. To initiate a dialog between the concerned communities, in this short note we wanted to give the distributed computing community a taste of the questions studied in parameterized model checking. More details can be found in our recent book [11].

While our initial goal in writing the book was to give a comprehensive survey on parameterized model checking, we quickly learned that the field is too lively for that. Consequently, we limited ourselves to decidability issues in concurrent systems of identical finite state processes. Indeed, our survey is just one in the area of parameterized model checking. Most recently, Delzanno [17] surveyed broadcast protocols. Before that Zuck and Pnueli [43] surveyed abstraction techniques. Regular model checking was surveyed by Abdulla [4], well-structured transition systems were surveyed by Finkel and Schnoebelen [29], and decidability in Petri nets was surveyed by Esparza [28].

Several parameterized model checking techniques have been implemented in tools such as BOOM [10], BYMC [12], T(O)RMC [33], TLV [38] and JTLV [37], Undip [3], MCMT [30], CHEAPS [32], MCMAS-P [35], and Cubicle [16]. Furthermore, there is a number of tools that implement more general forms of infinite-state model checking, in particular with support for integer-valued variables. With a suitable encoding, these tools can be used for parameterized model checking. Examples of such tools are ALV [42], BRAIN [40], FAST [9], MONA [22], and NUXMV [13]. Finally, the tool PARTY [31] uses decidability results surveyed in our book to synthesize parameterized systems from specifications.

Acknowledgments. We are grateful to Paul Attie, Giorgio Delzanno, Sayan Mitra, and Kedar Namjoshi for carefully reading a draft of our book, and providing detailed and constructive comments.

References

- [1] P. A. Abdulla, M. F. Atig, and O. Rezine. Verification of directed acyclic ad hoc networks. In *FORTE*, volume 7892 of *LNCS*, pages 193–208. Springer, 2013.
- [2] P. A. Abdulla, K. Čerāns, B. Jonsson, and Y.-K. Tsay. General decidability theorems for infinite-state systems. In *LICS*, pages 313 –321, 1996.
- [3] P. A. Abdulla, G. Delzanno, and A. Rezine. Approximated parameterized verification of infinite-state processes with global conditions. *Formal Methods in System Design*, 34(2):126–156, 2009.
- [4] P. A. Abdulla, B. Jonsson, M. Nilsson, and M. Saksena. A survey of regular model checking. In *CONCUR*, volume 3170 of *LNCS*, pages 35–48. Springer, 2004.
- [5] B. Aminof, S. Jacobs, A. Khalimov, and S. Rubin. Parameterized model checking of token-passing systems. In *VMCAI*, volume 8318 of *LNCS*, pages 262–281, Jan. 2014.
- [6] B. Aminof, T. Kotek, S. Rubin, F. Spegni, and H. Veith. Parameterized model checking of rendezvous systems. In *CONCUR*, volume 8704, pages 109–124. Springer, 2014.
- [7] B. Aminof, S. Rubin, and F. Zuleger. On the expressive power of communication primitives in parameterised systems. In M. Davis, A. Voronkov, A. McIver, and A. Fehnker, editors, *Logic for Programming, Artificial Intelligence, and Reasoning*, 2015.
- [8] H. Attiya and J. Welch. *Distributed Computing*. John Wiley & Sons, 2nd edition, 2004.
- [9] S. Bardin, A. Finkel, J. Leroux, and L. Petrucci. FAST: acceleration from theory to practice. *STTT*, 10(5):401–424, 2008.
- [10] G. Basler, M. Mazzucchi, T. Wahl, and D. Kroening. Symbolic counter abstraction for concurrent software. In *CAV*, volume 5643 of *LNCS*, pages 64–78. Springer, 2009.
- [11] R. Bloem, S. Jacobs, A. Khalimov, I. Konnov, S. Rubin, H. Veith, and J. Widder. *Decidability of Parameterized Verification*. Synthesis Lectures on Distributed Computing Theory. Morgan & Claypool Publishers, 2015.
- [12] ByMC. ByMC: Byzantine model checker, 2013. URL: <http://forsyte.tuwien.ac.at/software/bymc/>. Accessed: April, 2016.
- [13] R. Cavada, A. Cimatti, M. Dorigatti, A. Griggio, A. Mariotti, A. Micheli, S. Mover, M. Roveri, and S. Tonetta. The nuXmv symbolic model checker. In *CAV*, volume 8559 of *LNCS*, pages 334–342, 2014.
- [14] E. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 1999.
- [15] E. Clarke, M. Talupur, T. Touili, and H. Veith. Verification by network decomposition. In *CONCUR 2004*, volume 3170, pages 276–291, 2004.
- [16] S. Conchon, A. Goel, S. Krstic, A. Mebsout, and F. Zaïdi. Cubicle: A parallel smt-based model checker for parameterized systems - tool paper. In *CAV*, volume 7358 of *LNCS*, pages 718–724. Springer, 2012.

- [17] G. Delzanno. A unified view of parameterized verification of abstract models of broadcast communication. *International Journal on Software Tools for Technology Transfer*, pages 1–19, 2016.
- [18] G. Delzanno, J. Raskin, and L. Van Begin. Towards the automated verification of multithreaded Java programs. In *TACAS*, volume 2280 of *LNCS*, pages 173–187, 2002.
- [19] G. Delzanno, A. Sangnier, and G. Zavattaro. Parameterized verification of ad hoc networks. In *CONCUR*, volume 6269 of *LNCS*, pages 313–327, 2010.
- [20] G. Delzanno, A. Sangnier, and G. Zavattaro. On the power of cliques in the parameterized verification of ad hoc networks. In *FOSSACS*, volume 6604 of *LNCS*, pages 441–455. Springer, 2011.
- [21] G. Delzanno, A. Sangnier, and G. Zavattaro. Verification of ad hoc networks with node and communication failures. In *FORTE*, volume 7273 of *LNCS*, pages 235–250. Springer, 2012.
- [22] J. Elgaard, N. Klarlund, and A. Møller. MONA 1.x: new techniques for WS1S and WS2S. In *CAV*, volume 1427 of *LNCS*, pages 516–520. Springer, 1998.
- [23] E. A. Emerson and V. Kahlon. Reducing model checking of the many to the few. In *CADE*, volume 1831 of *LNCS*, pages 236–254. Springer Berlin Heidelberg, 2000.
- [24] E. A. Emerson and V. Kahlon. Exact and efficient verification of parameterized cache coherence protocols. In *CHARME*, volume 2860 of *LNCS*, pages 247–262. Springer, 2003.
- [25] E. A. Emerson and V. Kahlon. Model checking guarded protocols. In *LICS*, pages 361–370. IEEE, 2003.
- [26] E. A. Emerson and K. S. Namjoshi. Automatic verification of parameterized synchronous systems. In *CAV*, volume 1102 of *LNCS*, pages 87–98. Springer, 1996.
- [27] E. A. Emerson and K. S. Namjoshi. On reasoning about rings. *Int. J. Found. Comput. Sci.*, 14(4):527–550, 2003.
- [28] J. Esparza. Decidability and complexity of petri net problems - an introduction. In *In Lectures on Petri Nets I: Basic Models*, pages 374–428. Springer-Verlag, 1998.
- [29] A. Finkel and P. Schnoebelen. Well-structured transition systems everywhere! *Theoretical Computer Science*, 256(1–2):63–92, 2001.
- [30] S. Ghilardi and S. Ranise. Backward reachability of array-based systems by SMT solving: Termination and invariant synthesis. *Logical Methods in Computer Science*, 6(4), 2010.
- [31] A. Khalimov, S. Jacobs, and R. Bloem. PARTY parameterized synthesis of token rings. In *CAV*, volume 8044 of *LNCS*, pages 928–933. Springer, 2013.
- [32] I. V. Konnov and V. A. Zakharov. An invariant-based approach to the verification of asynchronous parameterized networks. *J. Symb. Comput.*, 45(11):1144–1162, 2010.
- [33] A. Legay. T(O)RMC: A tool for (omega)-regular model checking. In *CAV*, volume 5123 of *LNCS*, pages 548–551. Springer, 2008.

- [34] N. Lynch. *Distributed Algorithms*. Morgan Kaufman Publishers, Inc., San Francisco, USA, 1996.
- [35] MCMAS-P. VAS – Verification of autonomous systems, 2016. URL: <http://vas.doc.ic.ac.uk/software/extensions/>. Accessed: April 2016.
- [36] R. Milner. *Communication and concurrency*. PHI Series in computer science. Prentice Hall, 1989.
- [37] A. Pnueli, Y. Sa'ar, and L. D. Zuck. JTLV: A framework for developing verification algorithms. In *CAV*, volume 6174 of *LNCS*, pages 171–174. Springer, 2010.
- [38] A. Pnueli and E. Shahar. A platform for combining deductive with algorithmic verification. In *CAV*, volume 1102 of *LNCS*, pages 184–195. Springer, 1996.
- [39] K. V. S. Prasad. A calculus of broadcasting systems. *Sci. Comput. Program.*, 25(2–3):285–327, 1995.
- [40] T. Rybina and A. Voronkov. BRAIN : Backward reachability analysis with integers. In *AMAST*, volume 2422 of *LNCS*, pages 489–494. Springer, 2002.
- [41] I. Suzuki. Proving properties of a ring of finite-state machines. *Inf. Process. Lett.*, 28(4):213–214, July 1988.
- [42] T. Yavuz-Kahveci and T. Bultan. Action language verifier: an infinite-state model checker for reactive software specifications. *Formal Methods in System Design*, 35(3):325–367, 2009.
- [43] L. D. Zuck and A. Pnueli. Model checking and abstraction to the aid of parameterized systems (a survey). *Computer Languages, Systems & Structures*, 30(3-4):139–169, 2004.

Imperfect-Information Games and Generalized Planning

Giuseppe De Giacomo

SAPIENZA Università di Roma
Rome, Italy
degiacomo@dis.uniroma1.it

Aniello Murano, Sasha Rubin, Antonio Di Stasio

Università degli Studi di Napoli “Federico II”
Naples, Italy
first.last@unina.it

Abstract

We study a generalized form of planning under partial observability, in which we have multiple, possibly infinitely many, planning domains with the same actions and observations, and goals expressed over observations, which are possibly temporally extended. By building on work on two-player (non-probabilistic) games with imperfect information in the Formal Methods literature, we devise a general technique, generalizing the belief-state construction, to remove partial observability. This reduces the planning problem to a game of perfect information with a tight correspondence between plans and strategies. Then we instantiate the technique and solve some generalized-planning problems.

1 Introduction

Automated planning is a fundamental problem in Artificial Intelligence. Given a deterministic dynamic system with a single known initial state and a goal condition, automated planning consists of finding a sequences of actions (the plan) to be performed by agents in order to achieve the goal [M. Ghallab and Traverso, 2008]. The application of this notion in real-dynamic worlds is limited, in many situations, by three facts: i) the number of objects is neither small nor predetermined, ii) the agent is limited by its observations, iii) the agent wants to realize a goal that extends over time. For example, a preprogrammed driverless car cannot know in advance the number of obstacles it will enter in a road, or the positions of the other cars not in its view, though it wants to realize, among other goals, that every time it sees an obstacle it avoids it. This has inspired research in recent years on *generalized forms of planning* including conditional planning in partially observable domains [Levesque, 1996; Rintanen, 2004], planning with incomplete information for temporally extended goals [De Giacomo and Vardi, 1999; Bertoli and Pistore, 2004] and generalized planning for multiple domains or infinite domains [Levesque, 2005; Srivastava *et al.*, 2008; Bonet *et al.*, 2009; Hu and Levesque, 2010; Hu and De Giacomo, 2011; Srivastava *et al.*, 2011; Felli *et al.*, 2012; Srivastava *et al.*, 2015].

We use the following running example, taken from [Hu and Levesque, 2010], to illustrate a generalized form of planning:

Example 1 (Tree Chopping). The goal is to chop down a tree, and store the axe. The number of chops needed to fell the tree is unknown, but a look-action checks whether the tree is up or down. Intuitively, a plan solving this problem alternates looking and chopping until the tree is seen to be down, and then stores the axe. This scenario can be formalized as a partially-observable planning problem on a single infinite domain (Example 2, Figure 1), or on the disjoint union of infinitely many finite domains (Section 3).

The standard approach to solve planning under partial observability for *finite* domains is to reduce them to planning under complete observability. This is done by using the *belief-state construction* that removes partial observability and passes to the *belief-space* [Goldman and Boddy, 1996; Bertoli *et al.*, 2006]. The motivating problem of this work is to solve planning problems on infinite domains, and thus we are naturally lead to the problem of removing partial-observability from infinite domains.

In this paper we adopt the Formal Methods point of view and consider generalized planning as a game \mathbf{G} of imperfect information, i.e., where the player under control has partial observability. The game \mathbf{G} may be infinite, i.e., have infinitely many states.

Our technical contribution (Theorem 4.5) is a general sound and complete mathematical technique for removing imperfect information from a possibly infinite game \mathbf{G} to get a game \mathbf{G}^β , possibly infinite, of perfect information. Our method builds on the classic belief-state construction [Reif, 1984; Goldman and Boddy, 1996; Raskin *et al.*, 2007], also adopted in POMDPs [Kaelbling *et al.*, 1998; LaValle, 2006].¹ The classic belief-state construction fails for certain infinite games, see Example 3. We introduce a new component to the classic belief-state construction that isolates only those plays in the belief-space that correspond to plays in \mathbf{G} . This new component is necessary and sufficient to solve the general case and capture all infinite games \mathbf{G} .

We apply our technique to the decision problem that asks, given a game of imperfect information, if the player under control has a winning strategy (this corresponds to deciding if there is a plan for a given planning instance). We remark that we consider strategies and plans that may de-

¹However, our work considers nondeterminism rather than probability, and qualitative objectives rather than quantitative objectives.

pend on the history of the observations, not just the last observation. Besides showing how to solve the running Tree Chopping example, we report two cases. The first case is planning under partial observability for temporally extended goals expressed in LTL in finite domains (or a finite set of infinite domains sharing the same observations). This case generalizes well-known results in the AI literature [De Giacomo and Vardi, 1999; Rintanen, 2004; Bertoli *et al.*, 2006; Hu and De Giacomo, 2011; Felli *et al.*, 2012]. The second case involves infinite domains. Note that because game solving is undecidable for computable infinite games (simply code the configuration space of a Turing Machine), solving games with infinite domains requires further computability assumptions. We focus on games generated by pushdown automata; these are infinite games that recently attracted the interest of the AI community [Murano and Perelli, 2015; Chen *et al.*, 2016]. In particular these games have been solved assuming perfect information. By applying our technique, we extend their results to deal with imperfect information under the assumption that the stack remains observable (it is known that making the stack unobservable leads to undecidability [Azhar *et al.*, 2001]).

2 Generalized-Planning Games

In this section we define generalized-planning (GP) games, known as games of imperfect information in the Formal Methods literature [Raskin *et al.*, 2007], that capture many generalized forms of planning.

Informally, two players (agent and environment) play on a transition-system. Play proceeds in rounds. In each round, from the current state s of the transition-system, the agent observes $obs(s)$ (some information about the current state), and picks an action a from the set of actions Ac , and then the environment picks an element of $tr(s, a)$ (tr is the transition function of the transition-system) to become the new current state. Note that the players are asymmetric, i.e., the agent picks actions and the environment resolves non-determinism.

Notation. Write X^ω for the set of infinite sequences whose elements are from the set X , write X^* for the finite sequences, and X^+ for the finite non-empty sequences. If π is a finite sequence then $Last(\pi)$ denotes its last element. The positive integers are denoted \mathbb{N} , and $\mathbb{N}_0 := \mathbb{N} \cup \{0\}$.

Linear-temporal logic. We define LTL over a finite set of letters Σ .² The formulas of LTL (over Σ) are generated by the following grammar: $\varphi ::= x \mid \varphi \wedge \varphi \mid \neg \varphi \mid X \varphi \mid \varphi \vee \varphi$ where $x \in \Sigma$. We introduce the usual abbreviations for, e.g., \vee, F . Formulas of LTL (over Σ) are interpreted over infinite words $\alpha \in \Sigma^\omega$. Define the satisfaction relation \models as follows: i) $(\alpha, n) \models x$ iff $\alpha_n = x$; ii) $(\alpha, n) \models \varphi_1 \wedge \varphi_2$ iff $(\alpha, n) \models \varphi_i$ for $i = 1, 2$; iii) $(\alpha, n) \models \neg \varphi$ iff it is not the case that $(\alpha, n) \models \varphi$; iv) $(\alpha, n) \models X \varphi$ iff $(\alpha, n+1) \models \varphi$; v) $(\alpha, n) \models \varphi_1 \vee \varphi_2$ iff there exists $i \geq n$ such that $(\alpha, i) \models \varphi_2$ and for all $j \in [n, i)$, $(\alpha, j) \models \varphi_1$. Write $\alpha \models \varphi$ if $(\alpha, 0) \models \varphi$ and say that α satisfies the LTL formula φ .

²This is without loss of generality, since if LTL were defined over a set of atomic propositions AP we let $\Sigma = 2^{AP}$ and replace atoms $p \in AP$ by $\bigvee_{p \in x} x$ to get equivalent LTL formulas over Σ .

Arenas. An arena of imperfect information, or simply an arena, is a tuple $\mathbf{A} = (S, I, Ac, tr, Obs, obs)$, where S is a (possibly infinite) set of states, $I \subseteq S$ is the set of initial states, Ac is a finite set of actions, and $tr : S \times Ac \rightarrow 2^S \setminus \{\emptyset\}$ is the transition function, Obs is a (possibly infinite) set of observations, and $obs : S \rightarrow Obs$, the observation function, maps each state to an observation. We extend tr to sets of states: for $\emptyset \neq Q \subseteq S$, let $tr(Q, a)$ denote the set $\bigcup_{q \in Q} tr(q, a)$.

Sets of the form $obs^{-1}(x)$ for $x \in Obs$ are called observation sets. The set of all observation sets is denoted $ObsSet$. Non-empty subsets of observation sets are called belief-states. Informally, a belief-state is a subset of the states of the game that the play could be in after a given finite sequence of observations and actions.

Finite and finitely-branching. An arena is finite if S is finite, and infinite otherwise. An arena is finitely-branching if i) I is finite, and ii) for every s, a the cardinality of $tr(s, a)$ is finite. Clearly, being finite implies being finitely-branching.

Strategies. A play in \mathbf{A} is an infinite sequence $\pi = s_0 a_0 s_1 a_1 s_2 a_2 \dots$ such that $s_0 \in I$ and for all $i \in \mathbb{N}_0$, $s_{i+1} \in tr(s_i, a_i)$. A history $h = s_0 a_0 \dots s_{n-1} a_{n-1} s_n$ is a finite prefix of a play ending in a state. The set of plays is denoted $Ply(\mathbf{A})$, and the set of histories is denoted $Hist(\mathbf{A})$ (we drop \mathbf{A} when it is clear from the context). For a history or play $\pi = s_0 a_0 s_1 a_1 \dots$ write $obs(\pi)$ for the sequence $obs(s_0) a_0 obs(s_1) a_1 \dots$. A strategy (for the agent) is a function $\sigma : Hist(\mathbf{A}) \rightarrow Ac$. A strategy is observational if $obs(h) = obs(h')$ implies $\sigma(h) = \sigma(h')$. In Section 3 we will briefly mention an alternative (but essentially equivalent) definition of observational strategy, i.e., as a function $Obs^+ \rightarrow Ac$. We do not define strategies for the environment. A play $\pi = s_0 a_0 s_1 a_1 \dots$ is consistent with a strategy σ if for all $i \in \mathbb{N}$ we have that if $h \in Hist(\mathbf{A})$ is a prefix of π , say $h = s_0 a_0 \dots s_{n-1} a_{n-1} s_n$, then $\sigma(h) = a_{n+1}$.

GP Games. A generalized-planning (GP) game, is a tuple $\mathbf{G} = \langle \mathbf{A}, W \rangle$ where the winning objective $W \subseteq Obs^\omega$ is a set of infinite sequences of observation sets. A GP game with restriction is a tuple $\mathbf{G} = \langle \mathbf{A}, W, F \rangle$ where, in addition, $F \subseteq S^\omega$ is the restriction. Note that unlike the winning objective, the restriction need not be closed under observations. A GP game is finite (resp. finitely branching) if the arena \mathbf{A} is finite (resp. finitely branching).

Winning. A strategy σ is winning in $\mathbf{G} = \langle \mathbf{A}, W \rangle$ if for every play $\pi \in Ply(\mathbf{A})$ consistent with σ , we have that $obs(\pi) \in W$. Similarly, a strategy is winning in $\mathbf{G} = \langle \mathbf{A}, W, F \rangle$ if for every play $\pi \in Ply(\mathbf{A})$ consistent with σ , if $\pi \in F$ then $obs(\pi) \in W$. Note that a strategy is winning in $\langle \mathbf{A}, W, Ply(\mathbf{A}) \rangle$ if and only if it is winning in $\langle \mathbf{A}, W \rangle$.

Solving a GP game. A central decision problem is the following, called solving a GP game: given a (finite representation of a) GP game of imperfect information \mathbf{G} , decide if the agent has a winning observational-strategy.

GP games of perfect information. An arena/GP game has perfect information if $Obs = S$ and $obs(s) = s$ for all s . We thus suppress mentioning Obs and obs completely, e.g., we write $\mathbf{A} = (S, I, Ac, tr)$ and $W, F \subseteq S^\omega$. Note that in a GP game of perfect information every strategy is observational.

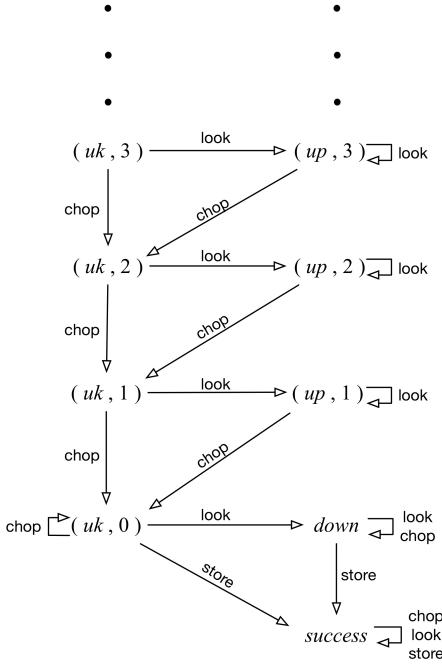


Figure 1: Part of the arena A_{chop} (missing edges go to the *failure* state). The numbers correspond to the number of chops required to fell the tree. The arena is not finitely-branching since it has infinitely many initial states $\{uk\} \times \mathbb{N}_0$.

Example 2 (continued). We formalize the tree-chopping planning problem. Define the GP game $G_{\text{chop}} = \langle A_{\text{chop}}, W \rangle$ where $A_{\text{chop}} = \langle S, I, Ac, \text{tr}, Obs, obs \rangle$, and:

- $S = \{down, success, failure\} \cup (\{uk\} \times \mathbb{N}_0) \cup (\{up\} \times \mathbb{N})$,
- $Ac = \{\text{chop}, \text{look}, \text{store}\}$, $I = \{uk\} \times \mathbb{N}$,
- tr is illustrated in Figure 1,
- $Obs = \{\text{DN}, \checkmark, \times, \text{UK}, \text{UP}\}$,
- obs maps $down \mapsto \text{DN}$, $(up, i) \mapsto \text{UP}$ for $i \in \mathbb{N}$, $(uk, i) \mapsto \text{UK}$ for $i \in \mathbb{N}_0$, $failure \mapsto \times$, and $success \mapsto \checkmark$, and
- the objective W is defined as $\alpha \in W$ iff $\alpha \models F \checkmark$.

The mentioned plan is formalized as the observational-strategy σ_{chop} that maps any history ending in (uk, i) to look (for $i \in \mathbb{N}_0$), (up, i) to chop (for $i \in \mathbb{N}$), $down$ to store, and all others arbitrarily (say to store).

Note: σ_{chop} is a winning strategy, i.e., no matter which initial state the environment chooses, the strategy ensures that the play (it is unique because the rest of the GP game is deterministic) reaches the state *success* having observation \checkmark .

3 Generalized-Planning Games and Generalized Forms of Planning

In this section we establish that generalized-planning (GP) games can model many different types of planning from the

AI literature, including a variety of generalized forms of planning:

1. planning on finite transition-systems, deterministic actions, actions with conditional effects, partially observable states, incomplete information on the initial state, and temporally extended goals [De Giacomo and Vardi, 1999];
2. planning under partial observability with finitely many state variables, nondeterministic actions, reachability goals, and partial observability [Rintanen, 2004];
3. planning on finite transition systems, nondeterministic actions, looking for strong plans (i.e., adversarial nondeterminism) [Bertoli *et al.*, 2006];
4. generalized planning, consisting of multiple (possibly infinitely many) related finite planning instances [Hu and Levesque, 2010; Hu and De Giacomo, 2011].

We discuss the latter in detail. Following [Hu and De Giacomo, 2011], a *generalized-planning problem* \mathfrak{P} is defined as a sequence of related classical planning problems. In our terminology, fix finite sets Ac, Obs and let \mathfrak{P} be a countable sequence G_1, G_2, \dots where each G_n is a finite GP game of the form $\langle S_n, \{\iota_n\}, Ac, \text{tr}_n, Obs, obs_n, W_n \rangle$. In [Hu and De Giacomo, 2011], a plan is an observational-strategy $p : Obs^+ \rightarrow Ac$, and a solution is a single plan that solves all of the GP games in the sequence. Now, we view \mathfrak{P} as a single infinite GP game as follows. Let $G_{\mathfrak{P}}$ denote the disjoint union of the GP games in \mathfrak{P} . Formally, $G_{\mathfrak{P}} = \langle S, I, Ac, \text{tr}, Obs, obs, W \rangle$ where

- $S = \{(s, n) : s \in S_n, n \in \mathbb{N}\}$,
- $I = \{(\iota_n, n) : n \in \mathbb{N}\}$,
- $\text{tr}((s, n), a) = \{(t, n) : t \in \text{tr}_n(s, a)\}$,
- $obs(s, n) = obs_n(s)$,
- $W = \bigcup_n W_n$.

Then: there is a correspondence between solutions for \mathfrak{P} and winning observational-strategies in $G_{\mathfrak{P}}$.

For example, consider the tree-chopping problem as formalized in [Hu and Levesque, 2010; Hu and De Giacomo, 2011]: there are infinitely many planning instances which are identical except for an integer parameter denoting the number of chops required to fell the tree. The objective for all instances is to fell the tree. Using the translation above we get a GP game with an infinite arena which resembles (and, in fact, can be transformed to) the GP game in Example 2.

4 Generalized Belief-State Construction

In this section we show how to remove imperfect information from generalized-planning (GP) games G . That is, we give a transformation of GP games of imperfect information G to GP games of perfect information G^{β} such that the agent has a winning observational-strategy in G if and only if the agent has a winning strategy in G^{β} . The translation is based on the classic belief-state construction [Reif, 1984; Raskin *et al.*, 2007]. Thus, we begin with a recap of that construction.

Belief-state Arena.³ Let $\mathbf{A} = (S, I, Ac, \text{tr}, \text{Obs}, obs)$ be an arena (not necessarily finite). Recall from Section 2 that observation sets are of the form $obs^{-1}(x)$ for $x \in \text{Obs}$, and are collectively denoted ObsSet. Define the arena of perfect information $\mathbf{A}^\beta = (S^\beta, I^\beta, Ac, \text{tr}^\beta)$ where,

- S^β is the set of belief-states, i.e., the non-empty subsets of the observation-sets,
- I^β consists of all belief-states of the form $I \cap X$ for $X \in \text{ObsSet}$,
- $\text{tr}^\beta(Q, a)$ consists of all belief-states of the form $\text{tr}(Q, a) \cap X$ for $X \in \text{ObsSet}$.

The idea is that $Q \in S^\beta$ represents a refinement of the observation set: the agent, knowing the structure of G ahead of time, and the sequence of observations so far in the game, may deduce that it is in fact in a state from Q which may be a strict subset of its corresponding observation set X .⁴

NB. Since \mathbf{A}^β is an arena, we can talk of its histories and plays. Although we defined S^β to be the set of all belief-states, only those belief-states that are reachable from I^β are relevant. Thus, overload notation and write S^β for the set of reachable belief-states, and \mathbf{A}^β for the corresponding arena. This notation has practical relevance since if \mathbf{A} is countable there are uncountably many belief-states; but in many cases only countably many (or, as in the running example, finitely many) reachable belief-states.

The intuition for the rest of this section is illustrated in the next example.

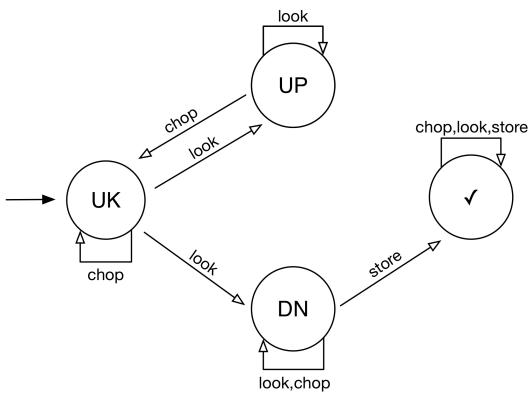


Figure 2: Part of the arena $\mathbf{A}^\beta_{\text{chop}}$ (missing edges go to the failure state). Each circle is a belief-state. The winning objective is $F \checkmark$, and the restriction is $\neg G F[\text{UK} \wedge X \text{ look} \wedge X \times \text{UP}]$.

Example 3 (continued). Figure 2 shows the arena $\mathbf{A}^\beta_{\text{chop}}$ corresponding to the arena from tree-chopping game \mathbf{G}_{chop} , i.e.,

³In the AI literature, this is sometimes called the *belief-space*.

⁴To illustrate simply, suppose there is a unique initial state s , and that it is observationally equivalent to other states. At the beginning of the game the agent can deduce that it must be in s . Thus, its initial belief-state is $\{s\}$ and not its observation-set $obs^{-1}(obs(s))$. This belief can (and, in general, must) be exploited if the agent is to win.

– S^β are the following belief-states: $\{(uk, n) \mid n \in \mathbb{N}_0\}$, denoted UK; $\{(up, n) \mid n \in \mathbb{N}\}$, denoted UP; $\{down\}$, denoted DN; $\{\text{success}\}$, denoted \checkmark ; and $\{\text{failure}\}$.

- I^β is the belief-state UK,
- and tr^β is shown in the figure.

Note that the agent does not have a winning strategy in the GP game with arena $\mathbf{A}^\beta_{\text{chop}}$ and winning condition $F \checkmark$. The informal reason is that the strategy σ_{chop} (which codifies “alternately look and chop until the tree is sensed to be down, and then store the axe”), which wins in G , does not work. The reason is that after every look the opponent can transition to UP (and never DN), resulting in the play $\rho = (\text{UK} \text{ look } \text{UP} \text{ chop})^\omega$, i.e., the repetition of $(\text{UK} \text{ look } \text{UP} \text{ chop})$ forever. Such a play of $\mathbf{A}^\beta_{\text{chop}}$ does not correspond to any play in \mathbf{A}_{chop} . This is a well known phenomena of the standard belief-set construction [Sardina *et al.*, 2006], which our construction overcomes by adding a restriction that removes from consideration plays such as ρ (as discussed in Example 5).

The following definition is central. It maps a history $h \in \text{Hist}(\mathbf{A})$ to the corresponding history $h^\beta \in \text{Hist}(\mathbf{A}^\beta)$ of belief-states.

Definition 4.1. For $h \in \text{Hist}(\mathbf{A})$ define $h^\beta \in \text{Hist}(\mathbf{A}^\beta)$ inductively as follows.

- For $s \in I$, define $s^\beta \in I^\beta$ to be $I \cap obs^{-1}(obs(s))$. In words, s^β is the set of initial states the GP game could be in given the observation $obs(s)$.
- If $h \in \text{Hist}(\mathbf{A})$, $a \in Ac$, $s \in S$, then $(has)^\beta := h^\beta a B$ where $B := \text{tr}(Last(h^\beta), a) \cap obs^{-1}(obs(s))$. In words, B is the set of possible states the GP game could be in given the observation sequence $obs(has)$.

In the same way, for $\pi \in \text{Ply}(\mathbf{A})$ define $\pi^\beta \in \text{Ply}(\mathbf{A}^\beta)$. Extend the map pointwise to sets of plays $P \subseteq \text{Ply}(\mathbf{A})$, i.e., define $P^\beta := \{\pi^\beta \in \text{Ply}(\mathbf{A}^\beta) \mid \pi \in P\}$. Finally, we give notation to the special case that $P = \text{Ply}(\mathbf{A})$: write $\text{Im}(\mathbf{A})$ for the set $\{\pi^\beta \mid \pi \in \text{Ply}(\mathbf{A})\}$, called the *image of \mathbf{A}* .

By definition, $\text{Im}(\mathbf{A}) \subseteq \text{Ply}(\mathbf{A}^\beta)$. However, the converse is not always true.

Example 4 (continued). There is a play of $\mathbf{A}^\beta_{\text{chop}}$ that is not in $\text{Im}(\mathbf{A}^\beta_{\text{chop}})$, e.g., $\rho = (uk \text{ look } up \text{ chop})^\omega$. Indeed, suppose $\pi^\beta = \rho$ and consider the sequence of counter values of π . Every look action establishes that the current counter value in π is positive (this is the meaning of the tree being *up*), but every chop action reduces the current counter value by one. This contradicts that counter values are always non-negative.

Remark 4.2. If \mathbf{A} is finitely-branching then $\text{Im}(\mathbf{A}) = \text{Ply}(\mathbf{A}^\beta)$. To see this, let ρ be a play in \mathbf{A}^β , and consider the forest whose nodes are the histories h of \mathbf{A} such that h^β is a prefix of ρ . Each tree in the forest is finitely branching (because \mathbf{A} is), and at least one tree in this forest is infinite. Thus, by König’s lemma, the tree has an infinite path π . But π is a play in \mathbf{A} and $\pi^\beta = \rho$.

Definition 4.3. For $\rho \in \text{Ply}(\mathbf{A}^\beta)$, say $\rho = B_0 a_0 B_1 a_1 \dots$, define $obs(\rho)$ to be the sequence $obs(q_0) a_0 obs(q_1) a_1 \dots$

where $q_i \in B_i$ for $i \in \mathbb{N}_0$ (this is well defined since, by definition of the state set S^β , each B_i is a subset of a unique observation-set).

The classic belief-state construction transforms $\langle A, W \rangle$ into $\langle A^\beta, W \rangle$. Example 3 shows that this transformation may not preserve the agent having a winning strategy if A is infinite. We now define the generalized belief-state construction and the main technical theorem of this work.

Definition 4.4. Let $G = \langle A, W \rangle$ be a GP game. Define $G^\beta = \langle A^\beta, W, \text{Im}(A) \rangle$, a GP game of perfect information with restriction. The restriction $\text{Im}(A) \subseteq \text{Ply}(A^\beta)$ is the image of $\text{Ply}(A)$ under the map $\pi \mapsto \pi^\beta$.

Theorem 4.5. Let A be a (possibly infinite) arena of imperfect information, A^β the corresponding belief-state arena of perfect information, and $\text{Im}(A) \subseteq \text{Ply}(A^\beta)$ the image of A . Then, for every winning objective W , the agent has a winning observational-strategy in the GP game $G = \langle A, W \rangle$ if and only if the agent has a winning strategy in the GP game $G^\beta = \langle A^\beta, W, \text{Im}(A) \rangle$. Moreover, if A is finitely-branching then $G^\beta = \langle A^\beta, W \rangle$.⁵

Proof. The second statement follows from the first statement and Remark 4.2. For the first statement, we first need some facts that immediately follow from Definition 4.1.

1. $(h_1)^\beta = (h_2)^\beta$ if and only if $\text{obs}(h_1) = \text{obs}(h_2)$.
2. For every $h \in \text{Hist}(A^\beta)$ that is also a prefix of π^β there is a history $h' \in \text{Hist}(A)$ that is also a prefix of π such that $(h')^\beta = h$. Also, for every $h' \in \text{Hist}(A)$ that is also a prefix of π there is a history $h \in \text{Hist}(A^\beta)$ that is also a prefix of π^β such that $(h')^\beta = h$.

Second, there is a natural correspondence between observational strategies of A and strategies of A^β .

- If σ is a strategy in A^β then define the strategy $\omega(\sigma)$ of A as mapping $h \in \text{Hist}(A)$ to $\sigma(h^\beta)$. Now, $\omega(\sigma)$ is observational by Fact 1. Also, if π is consistent with $\omega(\sigma)$ then π^β is consistent with σ . Indeed, let h be a history that is also a prefix of π^β . We need to show that $h\sigma(h)$ is a prefix of π^β . Suppose that $\sigma(h) = a$. Take prefix h' of h such that $(h')^\beta = h$ (Fact 2). Then $\omega(\sigma)(h') = \sigma((h')^\beta) = \sigma(h) = a$. Since π is assumed consistent with $\omega(\sigma)$, conclude that $h'a$ is a prefix of π . Thus ha is a prefix of π^β .
- If σ is an observational strategy in A then define the strategy $\kappa(\sigma)$ of A^β as mapping $h \in \text{Hist}(A^\beta)$ to $\sigma(h')$ where h' is any history such that $h'^\beta = h$. This is well-defined by (†) and the fact that σ is observational. Also, if ρ is consistent with $\kappa(\sigma)$, then every π with $\pi^\beta = \rho$ (if there are any) is consistent with σ . Indeed, let h' be a history of π and take a prefix h of π^β such that $(h')^\beta = h$ (Fact 2). Then $\kappa(\sigma)(h) = \sigma(h')$, call this action a . But π^β is assumed consistent with $\kappa(\sigma)$, and thus ha is a prefix of π^β . Thus $h'a$ is a prefix of π .

We now put everything together and show that the agent has a winning observational-strategy in G iff the agent has a winning strategy in G^β .

⁵The case that A is finite appears in [Raskin *et al.*, 2007].

Suppose σ is a winning strategy in G^β . Let $\pi \in \text{Ply}(A)$ be consistent with the observational strategy $\omega(\sigma)$ of G . Then $\pi^\beta \in \text{Im}(A)$ is consistent with σ . But σ is assumed to be winning, thus $\text{obs}(\pi^\beta) \in W$. But $\text{obs}(\pi) = \text{obs}(\pi^\beta)$. Conclude that $\omega(\sigma)$ is a winning strategy in G .

Conversely, suppose σ is a winning strategy in G . Let $\rho \in \text{Im}(A)$ be consistent with the strategy $\kappa(\sigma)$ of G^β , and take $\pi \in \text{Ply}(A)$ be such that $\pi^\beta = \rho$ (such a π exists since we assumed $\rho \in \text{Im}(A)$). Then π is consistent with σ . But σ is assumed to be winning, thus $\text{obs}(\pi) \in W$. But $\text{obs}(\rho) = \text{obs}(\pi^\beta) = \text{obs}(\pi)$. Conclude that $\kappa(\sigma)$ is a winning strategy in G^β . \square

Remark 4.6. The proof of Theorem 4.5 actually shows how to transform strategies between the GP games, i.e., $\sigma \mapsto \omega(\sigma)$ and $\sigma \mapsto \kappa(\sigma)$, and moreover, these transformations are inverses of each other.

We end with our running example:

Example 5 (Continued). Solving G_{chop} (Figure 1) is equivalent to solving the finite GP game G_{chop}^β of perfect information, i.e., $\langle A_{\text{chop}}^\beta, W, \text{Im}(A_{\text{chop}}) \rangle$, where the arena A is shown in Figure 2. To solve this we should understand the structure of $\text{Im}(A_{\text{chop}})$. It is not hard to see that a play $\rho \in \text{Ply}(A_{\text{chop}}^\beta)$ is in $\text{Im}(A_{\text{chop}})$ if and only if it contains only finitely many infixes of the form “UK look UP”. This property is expressible in LTL by the formula $\neg G F[\text{UK} \wedge X \text{look} \wedge XX \text{UP}]$. Thus we can apply the algorithm for solving finite games of perfect information with LTL objectives (see, e.g., [Pnueli and Rosner, 1989; de Alfaro *et al.*, 2001]) to solve G_{chop}^β , and thus the original GP game G_{chop} .

5 Application of the Construction

We now show how to use generalized-planning (GP) games and our generalized belief-state construction to obtain effective planning procedures for sophisticated problems. For the rest of this section we assume Obs is finite (A may be infinite) so that we can consider LTL temporally extended goals over the alphabet Obs. For instance, LTL formulas specify persistent surveillance missions such as “get items from region A, drop items at region B, infinitely often, and always avoid region C”.

Definition 5.1. Let φ be an LTL formula over $\text{Obs} \times \text{Ac}$. For an arena A , define $[[\varphi]] = \{\pi \in \text{Ply}(A) \mid \text{obs}(\pi) \models \varphi\}$.

The following is immediate from Theorem 4.5 and the fact that solving finite LTL games of perfect information is decidable [Pnueli and Rosner, 1989; de Alfaro *et al.*, 2001]:

Theorem 5.2. Let $G = \langle A, [[\varphi]] \rangle$ be a GP game with a finite arena (possibly obtained as the disjoint union of several arenas sharing the same observations), and φ be an LTL winning objective. Then solving G can be reduced to solving the finite GP game $G^\beta = \langle A^\beta, [[\varphi]] \rangle$ of perfect information, which is decidable.

Although we defined winning objectives to be observable, one may prefer general winning conditions, i.e., $W \subseteq S^\omega$. In this case, for finite arenas there is a translation from parity-objectives to observable parity-objectives [Chatterjee

and Doyen, 2010]; moreover, for reachability objectives, a plan reaches a goal $T \subseteq S$ iff it reaches a belief-state $B \subseteq T$ [Bertoli *et al.*, 2006].

Next we look a case where the arena is actually infinite. Recently, the AI community has considered games generated by pushdown-automata [Murano and Perelli, 2015; Chen *et al.*, 2016]. However, the games considered are of perfect information and cannot express generalized-planning problems or planning under partial observability. In contrast, our techniques can solve these planning problems on pushdown domains assuming that the stack is not hidden (we remark that if the stack is hidden, then game-solving becomes undecidable [Azhar *et al.*, 2001]):

Theorem 5.3. *Let $\mathbf{G} = \langle \mathbf{A}, [[\varphi]] \rangle$ be a GP game with a pushdown-arena with observable stack, and φ is an LTL formula. Then solving \mathbf{G} can be reduced to solving $\mathbf{G}^\beta = \langle \mathbf{A}^\beta, [[\varphi]] \rangle$, a GP game with pushdown-arena with perfect information, which is decidable.*

Proof. Let P be a pushdown-automaton with states Q , initial state q_0 , finite input alphabet Ac , and finite stack alphabet Γ . We call elements of Γ^* stacks, and denote the empty stack by ϵ . Also, fix an observation function on the states, i.e., $f : Q \rightarrow \text{Obs}$ for some set Obs (we do not introduce notation for the transition function of P). A pushdown-arena $\mathbf{A}_P = \langle S, I, Ac, \text{tr}, \text{Obs}, obs \rangle$ is generated by P as follows: the set of states S is the set of configurations of P , i.e., pairs (q, γ) where $q \in Q$ and $\gamma \in \Gamma^*$ is a stack-content of P ; the initial state of \mathbf{A} is the initial configuration, i.e., $I = \{(q_0, \epsilon)\}$; the transition function of \mathbf{A} is defined as $\text{tr}((q, \gamma), a) = (q', \gamma')$ if P can move in one step from state q and stack content γ to state q' and stack content γ' by consuming the input letter a ; the observation function obs maps a configuration (q, γ) to $f(q)$ (i.e., this formalizes the statement that the stack is observable). Observe now that: (1) the GP game \mathbf{A} is finitely-branching; (2) the GP game \mathbf{A}^β is generated by a pushdown automaton (its states are subsets of Q). Thus we can apply Theorem 4.5 to reduce solving \mathbf{A} , a GP-game with imperfect information and pushdown arena, to \mathbf{A}^β , a GP-game with perfect information and pushdown arena. The latter is decidable [Walukiewicz, 2001]. \square

6 Related work in Formal Methods

Games of imperfect information on *finite* arenas have been studied extensively. Reachability winning-objectives were studied in [Reif, 1984] from a complexity point of view: certain games were shown to be universal in the sense that they are the hardest games of imperfect information, and optimal decision procedures were given. More generally, ω -regular winning-objectives were studied in [Raskin *et al.*, 2007], and symbolic algorithms were given (also for the case of randomized strategies).

To solve (imperfect-information) games on infinite arenas one needs a finite-representation of the infinite arena. One canonical way to generate infinite arenas is by parametric means. In this line, [Jacobs and Bloem, 2014] study the synthesis problem for distributed architectures with a parametric number of finite-state components. They leverage re-

sults from the Formal Methods literature that say that for certain types of token-passing systems there is a cutoff [Emerson and Namjoshi, 1995], i.e., an upper bound on the number of components one needs to consider in order to synthesize a protocol for any number of components. Another way to generate infinite arenas is as configuration spaces of pushdown automata. These are important in analysis of software because they capture the flow of procedure calls and returns in reactive programs. Module-checking pushdown systems of imperfect information [Bozzelli *et al.*, 2010; Aminof *et al.*, 2013] can be thought of as games in which the environment plays non-deterministic strategies. Although undecidable, by not hiding the stack (cf. Theorem 5.3) decidability of module-checking is regained.

Finally, we note that synthesis of distributed systems has been studied in the Formal Methods literature using the techniques of games, starting with [Pnueli and Rosner, 1990]. Such problems can be cast as multi-player games of imperfect information, and logics such as ATL with knowledge can be used to reason about strategies in these games. However, even for three players, finite arenas, and reachability goals, the synthesis problem (and the corresponding model checking problem for ATL K) is undecidable [Dima and Tiplea, 2011].

7 Critical Evaluation and Conclusions

Although our technique for removing partial observability is sound and complete, it is, necessarily, not algorithmic: indeed, no algorithm can always remove partial observability from computable infinite domains and result in a solvable planning problem (e.g, one with a finite domain).⁶

The main avenue for future technical work is to establish natural classes of generalized-planning problems that can be solved algorithmically. We believe the methodology of this paper will be central to this endeavor. Indeed, as we showed in Section 5, we can identify $\text{Im}(\mathbf{A})$ in a number of cases. We conjecture that one can do the same for all of the one-dimensional planning problems of [Hu and Levesque, 2010; Hu and De Giacomo, 2011].

The framework presented in this paper is non-probabilistic, but extending it with probabilities and utilities associated to agent choices [Kaelbling *et al.*, 1998; LaValle, 2006; Bonet and Geffner, 2009; Geffner and Bonet, 2013] is of great interest. In particular, POMDPs with temporally-extended winning objectives (e.g., LTL, Büchi, parity) have been studied for finite domains [Chatterjee *et al.*, 2010]. We leave for future work the problem of dealing with such POMDPs over infinite domains.

Acknowledgments

We thank the reviewers for their constructive comments. This research was partially supported by the Sapienza project “Immersive Cognitive Environments”. Aniello Murano was supported in part by GNCS 2016 project: Logica, Automi e Giochi per Sistemi Auto-adattivi. Sasha Rubin is a Marie Curie fellow of the Istituto Nazionale di Alta Matematica.

⁶In fact, there is no algorithm solving games of perfect observation on computable domains with reachability objectives.

References

- [Aminof *et al.*, 2013] B. Aminof, A. Legay, A. Murano, O. Serre, and M. Y. Vardi. Pushdown module checking with imperfect information. *Inf. Comput.*, 223, 2013.
- [Azhar *et al.*, 2001] S. Azhar, G. Peterson, and J. Reif. Lower bounds for multiplayer non-cooperative games of incomplete information. *J. Comp. Math. Appl.*, 41, 2001.
- [Bertoli and Pistore, 2004] P. Bertoli and M. Pistore. Planning with extended goals and partial observability. In *Proc. of ICAPS 2004*, 2004.
- [Bertoli *et al.*, 2006] P. Bertoli, A. Cimatti, M. Roveri, and P. Traverso. Strong planning under partial observability. *Artif. Intell.*, 170(4-5), 2006.
- [Bonet and Geffner, 2009] B. Bonet and H. Geffner. Solving POMDPs: Rtdp-bel vs. point-based algorithms. In *Proc. of IJCAI 2009*, 2009.
- [Bonet *et al.*, 2009] B. Bonet, H. Palacios, and H. Geffner. Automatic derivation of memoryless policies and finite-state controllers using classical planners. In *Proc. of ICAPS 2009*, 2009.
- [Bozzelli *et al.*, 2010] L. Bozzelli, A. Murano, and A. Peron. Pushdown module checking. *FMSD*, 36(1), 2010.
- [Chatterjee and Doyen, 2010] K. Chatterjee and L. Doyen. The complexity of partial-observation parity games. In *Proc. of LPAR 2010*, 2010.
- [Chatterjee *et al.*, 2010] K. Chatterjee, L. Doyen, and T.A. Henzinger. Qualitative analysis of partially-observable markov decision processes. In *Proc. of MFCS*, 2010.
- [Chen *et al.*, 2016] T. Chen, F. Song, and Z. Wu. Global model checking on pushdown multi-agent systems. In *Proc. of AAAI 2016*, 2016.
- [de Alfaro *et al.*, 2001] L. de Alfaro, T. A. Henzinger, and R. Majumdar. From verification to control: Dynamic programs for omega-regular objectives. In *Proc. of LICS 2001*, 2001.
- [De Giacomo and Vardi, 1999] G. De Giacomo and M. Y. Vardi. Automata-theoretic approach to planning for temporally extended goals. In *Proc. of ECP 1999*, 1999.
- [Dima and Tiplea, 2011] C. Dima and F. L. Tiplea. Model-checking ATL under imperfect information and perfect recall semantics is undecidable. *CoR*, abs/1102.4225, 2011.
- [Emerson and Namjoshi, 1995] E. A. Emerson and K. S. Namjoshi. Reasoning about rings. In *Proc. of POPL 1995*, 1995.
- [Felli *et al.*, 2012] P. Felli, G. De Giacomo, and A. Lomuscio. Synthesizing agent protocols from LTL specifications against multiple partially-observable environments. In *Proc. of KR 2012*, 2012.
- [Geffner and Bonet, 2013] H. Geffner and B. Bonet. *A Concise Introduction to Models and Methods for Automated Planning*. Morgan & Claypool, 2013.
- [Goldman and Boddy, 1996] R. P. Goldman and M. S. Boddy. Expressive planning and explicit knowledge. In *Proc. of AIPS 1996*, 1996.
- [Hu and De Giacomo, 2011] Y. Hu and G. De Giacomo. Generalized planning: Synthesizing plans that work for multiple environments. In *Proc. of IJCAI 2011*, 2011.
- [Hu and Levesque, 2010] Y. Hu and H. J. Levesque. A correctness result for reasoning about one-dimensional planning problems. In *Proc. of KR 2010*, 2010.
- [Jacobs and Bloem, 2014] S. Jacobs and R. Bloem. Parameterized synthesis. *LMCS*, 10(1), 2014.
- [Kaelbling *et al.*, 1998] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra. Planning and acting in partially observable stochastic domains. *Artif. Intell.*, 101(1-2), 1998.
- [LaValle, 2006] S. M. LaValle. *Planning algorithms*. Cambridge, 2006.
- [Levesque, 1996] H. J. Levesque. What is planning in the presence of sensing. In *Proc. of AAAI 1996*, 1996.
- [Levesque, 2005] H. Levesque. Planning with loops. In *Proc. of IJCAI 2005*, 2005.
- [M. Ghallab and Traverso, 2008] D. Nau M. Ghallab and P. Traverso. *Automated Planning: Theory & Practice*. Elsevier, 2008.
- [Murano and Perelli, 2015] A. Murano and G. Perelli. Pushdown multi-agent system verification. In *Proc. of IJCAI 2015*, 2015.
- [Pnueli and Rosner, 1989] A. Pnueli and R. Rosner. On the synthesis of an asynchronous reactive module. In *Proc. of ICALP 1989*, 1989.
- [Pnueli and Rosner, 1990] A. Pnueli and R. Rosner. Distributed reactive systems are hard to synthesize. In *Proc. of STOC 1990*, 1990.
- [Raskin *et al.*, 2007] J. Raskin, K. Chatterjee, L. Doyen, and T. A. Henzinger. Algorithms for omega-regular games with imperfect information. *LMCS*, 3(3), 2007.
- [Reif, 1984] J. H. Reif. The complexity of two-player games of incomplete information. *JCSS*, 29(2), 1984.
- [Rintanen, 2004] J. Rintanen. Complexity of planning with partial observability. In *Proc. of ICAPS 2004*, 2004.
- [Sardiña *et al.*, 2006] S. Sardiña, G. De Giacomo, Y. Lespérance, and H. J. Levesque. On the limits of planning over belief states under strict uncertainty. In *Proc. of KR 2016*, 2006.
- [Srivastava *et al.*, 2008] S. Srivastava, N. Immerman, and S. Zilberstein. Learning generalized plans using abstract counting. In *Proc. of AAAI 2008*, 2008.
- [Srivastava *et al.*, 2011] S. Srivastava, N. Immerman, and S. Zilberstein. A new representation and associated algorithms for generalized planning. *Artif. Intell.*, 175(2), 2011.
- [Srivastava *et al.*, 2015] S. Srivastava, S. Zilberstein, A. Gupta, P. Abbeel, and S. J. Russell. Tractability of planning with loops. In *Proc. of AAAI 2015*, 2015.
- [Walukiewicz, 2001] I. Walukiewicz. Pushdown processes: Games and model-checking. *Inf. Comput.*, 164(2), 2001.

Verification of Multi-agent Systems with Imperfect Information and Public Actions

F. Belardinelli

Laboratoire IBISC, UEVE – IRIT Toulouse
belardinelli@ibisc.fr

A. Lomuscio

Department of Computing
Imperial College London
a.lomuscio@imperial.ac.uk

A. Murano and S. Rubin

DIETI
Università degli Studi di Napoli
murano@na.infn.it
rubin@unina.it

ABSTRACT

We analyse the verification problem for synchronous, perfect recall multi-agent systems with imperfect information against a specification language that includes strategic and epistemic operators. While the verification problem is undecidable, we show that if the agents' actions are public, then verification is 2EXPTIME-complete. To illustrate the formal framework we consider two epistemic and strategic puzzles with imperfect information and public actions: the muddy children puzzle and the classic game of battleships.

1. INTRODUCTION

Synchronous, perfect-recall multi-agent systems (MAS) are an important class of MAS that can be used to model a wide variety of scenarios including communication protocols, security protocols and games [8]. Reasoning about the knowledge and the strategic ability of agents in these systems remains of particular importance. Traditionally, epistemic logic [8] has been used to express the states of knowledge of the agents, whereas ATL has provided a basis for the agents' strategic abilities [1]. ATL and epistemic logic have been combined in a number of ways to obtain specification languages capable of expressing both concepts (see below). A popular method for establishing properties of MAS is verification via model checking [4].

However, verifying synchronous, perfect recall MAS under incomplete information against specifications in ATL is undecidable [1, 6] (hence it remains undecidable when epistemic modalities are added); it is therefore of interest to identify cases in which reasoning about MAS is decidable. These restrictions typically take three forms: restricting the syntax of the logic (e.g., by removing strategic abilities and consider, instead, $LTL_{\mathbb{K}}$, the extension of LTL with individual-knowledge operators, as in [32]), restricting the semantics (e.g., by requiring strategy quantifiers to vary over memoryless-strategies [31]), or by restricting the class of MAS under consideration. In this paper we follow the third option.

Contribution. We identify a class of imperfect-information concurrent game structures (iCGS) that we call public-action iCGS (PA-iCGS). In contrast to general iCGS [6], we prove

that model-checking the full logic $ATL_{\mathbb{K},C,D}^*$ on PA-iCGS is decidable, specifically 2EXPTIME-complete. Thus, the joint complexity of model-checking is the same as that of ATL^* with perfect information [1]. Moreover, we show that the class models MAS in which agents have imperfect information, synchronous perfect recall, and whose actions are public, i.e., all actions are visible to all agents. As we explain, the class PA-iCGS captures games of imperfect information in which the agents have uncertainty about the initial configuration but all moves are observable to all agents. This has applications to, among others, games (e.g., Bridge, Poker, Battleships, etc.), fair division protocols (e.g., classic cake cutting algorithms), selected broadcast protocols [33], blackboard systems in which a public database is read and written by agents [26], auctions and auction-based mechanisms [7].

The rest of the paper is organised as follows. In the remainder of this section we discuss related work. In Section 2 we define iCGS with public actions and the logic $ATL_{\mathbb{K},C,D}^*$, that we will use as specification language and illustrate the formalism. In Section 3 we present the main result of the paper, i.e., we show the decidability of the verification problem, by means of an automata-theoretic approach, and analyse the resulting complexity. In Section 4 we compare our approach to that of Broadcast Environments [33]. We conclude in Section 5.

Related Work. In order to reason formally about multi-agent systems, temporal logics such as LTL, CTL, CTL* have been extended with strategy quantifiers [1] and epistemic modalities [14]. The extended syntax has been combined with a number of different assumptions on the underlying MAS: perfect vs. imperfect information, perfect vs. imperfect recall, state-based vs. history-based semantics [1, 14, 10, 31, 15, 5, 11, 25].

Assuming imperfect information and perfect recall, as we do in this paper, often results in intractable model-checking. For instance, the model-checking problem for ATL in this setting is undecidable [6], as is the model-checking problem for the extension $LTL_{\mathbb{K},C}$ of LTL with epistemic operators, including common knowledge [32]. Given this difficulty, finding decidable or tractable fragments remains of interest.

As expected, restricting the logic lowers the complexity. We list some notable examples: the model-checking problem for $LTL_{\mathbb{K}}$ with only individual knowledge is non-elementary complete [32], model-checking ATL with only “communicating coalitions” (i.e., coalitions use their distributed knowledge instead of their individual knowledge) is decidable and non-elementary [5, 11]; and, model-checking ATL in which all coalitions operate with a single indistinguishability relation

tion reduces ATL to its singleton-coalition fragment [17].

Also, restricting the class of structures (iCGS) over which these languages are interpreted lowers the complexity. Such restrictions typically take one of two forms: i) on the observation or information sets of the agents; ii) on the architectures that govern communication. Notable examples of i) may require that: all agents have the same observation sets [21]; that the information sets form a hierarchy [27], or that, over time, they infinitely often form a hierarchy [2]. A notable example of ii) are characterisations of the architectures for which distributed synthesis is decidable [9, 30], thus generalising earlier results on linear architectures [27, 18].

More closely related to the present contribution are broadcast environments (which restrict the underlying iCGS) and that can capture epistemic puzzles and games of imperfect information such as Bridge [33]. The most relevant result for broadcast environments is that synthesis of linear-temporal logic with individual-knowledge operators is decidable [33]. Not only can our language express this synthesis problem, but it is strictly more expressive, as it can alternate strategic quantifiers mentioning overlapping coalitions. A detailed discussion of the significance of [33] is given in Section 4.

Actions that constitute public announcements have been studied in depth (Dynamic Epistemic Logic, Public Announcement Logic, epistemic protocols [34]). However, this line of research differs semantically and syntactically from our work. In particular, in these works modal operators are model transformers, and coalitions are not explicitly named in the language.

2. GAMES WITH PUBLIC ACTIONS AND STRATEGIC-EPISTEMIC LOGIC

In this section we define the game model and the logic. The model is the subclass of imperfect information concurrent game structures (iCGS) that only have public actions (PA-iCGS). The logic is $\text{ATL}_{\mathbb{K}, \mathbb{C}, \mathbb{D}}^*$, an extension of Alternating Time Temporal Logic (ATL*) which includes strategic operators ($\langle\!\langle A \rangle\!\rangle$ for $A \subseteq Ag$) as well as epistemic operators for individual-knowledge (\mathbb{K}_a for $a \in Ag$), common-knowledge (\mathbb{C}_A for $A \subseteq Ag$), and distributed-knowledge (\mathbb{D}_A for $A \subseteq Ag$).

Notation. For an infinite or non-empty finite sequence $u \in X^\omega \cup X^+$ write u_i for the i th element of u , i.e., $u = u_0 u_1 \dots$. The empty sequence is denoted ϵ . The length of a finite sequence $u \in X^*$ is denoted $|u|$, its last (resp. first) element is denoted $last(u)$ (resp. $first(u)$). Note that $last(\epsilon) = first(\epsilon) = \epsilon$. For $i < |u|$ write $u_{\leq i}$ for the prefix $u_0 \dots u_i$. For a vector $v \in \prod_i X_i$ we denote the i -th co-ordinate of v by $v(i)$. In particular, for $F \in \prod_i (X_i)^Y$ we may write $F(i) \in X^Y$ and $F(i)(y) \in X_i$.

2.1 iCGS with only Public Actions

We begin with the standard definition of imperfect information concurrent game structures [3, 6].

DEFINITION 1 (iCGS). *An imperfect information concurrent game structure (iCGS) is a tuple*

$$S = \langle Ag, AP, \{Act_a\}_{a \in Ag}, S, S_0, \delta, \{\sim_a\}_{a \in Ag}, \lambda \rangle$$

where:

- Ag is the finite non-empty set of agent names;

- AP is the finite non-empty set of atomic propositions;
- Act_a is the finite non-empty set of actions for $a \in Ag$;
- S is the finite non-empty set of states;
- $S_0 \subseteq S$ is the non-empty set of initial states;
- $\delta : S \times ACT \rightarrow S$ is the transition function, where the set ACT of joint-actions is the set of all functions $J : Ag \rightarrow \bigcup_a Act_a$ such that $J(a) \in Act_a$. The transition function assigns to every state s and joint-action J , a successor state $\delta(s, J)$;
- $\sim_a \subseteq S^2$ is the indistinguishability relation for agent a , which is an equivalence relation; the equivalence class $[s]_a$ of $s \in S$ under \sim_a is called the observation set of agent a ;
- $\lambda : AP \rightarrow 2^S$ is the labeling function that assigns to each atom p the set of states $\lambda(p)$ in which p holds.

Perfect-information is treated as a special case:

DEFINITION 2 (PERFECT-INFORMATION). *A concurrent game structure (CGS) is an iCGS for which $\sim_a = \{(s, s) : s \in S\}$ for all $a \in Ag$.*

We now give a brief and to-the-point definition of what it means for an iCGS to only have public actions, i.e., all actions are visible to all agents. This determines a subclass of iCGS that we call PA-iCGS.

DEFINITION 3 (PA-iCGS). *An iCGS only has public actions if for every agent $a \in Ag$, states $s, s' \in S$, and joint actions $J, J' \in ACT$, if $J \neq J'$ and $s \sim_a s'$ then $\delta(s, J) \not\sim_a \delta(s', J')$. We write PA-iCGS for the class of iCGS that only have public actions.*

This definition says that if an agent a cannot distinguish between two states, but different joint actions are performed in each of these states (because, for instance, some other agent can distinguish them), then the agent can distinguish between the resulting successor states.

One way to generate an iCGS only having public actions is to ensure that i) the state records the last joint-action played, thus S is of the form $T \times (ACT \cup \{\epsilon\})$, where ϵ refers to the situation that no actions have yet been played, and ii) the indistinguishability relations \sim_a satisfy that if $(t, J) \sim_a (t', J')$ then $J = J'$. Similar conditions have been considered in the literature, e.g., *recording contexts* in [8].

In the remainder of this section we define what it means for an agent to have *synchronous perfect-recall* [8].

Synchronous perfect-recall under imperfect information. A path in S is a non-empty infinite or finite sequence $\pi_0 \pi_1 \dots \in S^\omega \cup S^+$ such that for all i there exists a joint-action $J(i) \in ACT$ such that $\pi_{i+1} \in \delta(\pi_i, J(i))$. Paths that start with initial states are called *histories* if they are finite and *computations* if they are infinite. The set of computations in S is written $\text{comp}(S)$, and the set of computations in S that start with history h is written $\text{comp}(S, h)$. We define $\text{hist}(S)$ and $\text{hist}(S, h)$ similarly.

We use the following notation: if \sim is a binary relation on S we define the extension of \sim to histories as the binary relation \equiv on $\text{hist}(S)$ defined by $h \equiv h'$ iff $|h| = |h'|$ (i.e., synchronicity) and $h_j \sim h'_j$ for all $j \leq |h|$ (i.e., perfect recall).

We give three particular instantiations. If \sim_a is the indistinguishability relation for agent a , then we say that two histories h, h' are *indistinguishable to agent a* , if $h \equiv_a h'$. For $A \subseteq Ag$, let $\sim_A^C = (\cup_{a \in A} \sim_a)^*$, where $*$ denotes the reflexive transitive closure (wrt. composition of relations), and its extension to histories is denoted \equiv_A^C . For $A \subseteq Ag$, let $\sim_A^D = \cap_{a \in A} \sim_a$, and its extension to histories is denoted \equiv_A^D .

Strategies. A *deterministic memoryfull strategy*, or simply a *strategy*, for agent a is a function $\sigma_a : \text{hist}(S) \rightarrow \text{Act}_a$. A strategy σ_a is *uniform* if for all $h \equiv_a h'$, we have $\sigma(h) = \sigma(h')$. The set of uniform strategies is denoted $\Sigma(S)$. All strategies in the rest of the paper are uniform (although sometimes we will stress this fact and write “uniform strategy”).

For $A \subseteq Ag$, let $\sigma_A : A \rightarrow \Sigma(S)$ denote a function that associates a uniform strategy σ_a with each agent $a \in A$. We write $\sigma_A(a) = \sigma_a$, and call σ_A a *joint strategy*.

For $h \in \text{hist}(S)$ write $\text{out}(S, h, \sigma_A)$, called the *outcomes of σ_A from h* , for the set of computations $\pi \in \text{comp}(S, h)$ such that π is consistent with σ_A , that is, $\pi \in \text{out}(S, h, \sigma_A)$ iff (i) $\pi_{\leq |h|-1} = h$; (ii) for every position $i \geq |h|$, there exists a joint-action $J_i \in \text{ACT}$ such that $\pi_{i+1} \in \delta(\pi_i, J_i)$, and for every $a \in A$, $J_i(a) = \sigma_A(a)(\pi_{\leq i})$. We may drop S and write simply $\text{out}(h, \sigma_A)$. Notice that, if $A = \emptyset$, then $\text{out}(h, \sigma_A)$ is the set of all paths starting with h (this is because σ_A is the empty function and (ii) above places no additional restrictions on the computations).

2.2 The Logic $\text{ATL}_{K,C,D}^*$

In this section we define the logic $\text{ATL}_{K,C,D}^*$. Its syntax has been called ATEL^* (cf. [14]), and we interpret it on iCGS with history-based semantics and imperfect information.

Syntax. Fix a finite set of *atomic propositions (atoms)* AP and a finite set of *agents* Ag . The *history (φ) and path (ψ) formulas over AP and Ag* are built using the following grammar:

$$\begin{aligned}\varphi &::= p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \mathbb{K}_a \varphi \mid \mathbb{C}_A \varphi \mid \langle\langle A \rangle\rangle \psi \\ \psi &::= \varphi \mid \neg\psi \mid \psi \wedge \psi \mid \mathbf{X} \psi \mid \psi \mathbf{U} \psi\end{aligned}$$

where $p \in AP$, $a \in Ag$, and $A \subseteq Ag$.

The class of $\text{ATL}_{K,C,D}^*$ formulas is the set of history formulas generated by the grammar. The *temporal operators* are \mathbf{X} (read “next”) and \mathbf{U} (read “until”). The *strategy quantifier* is $\langle\langle A \rangle\rangle$ (“the agents in A can enforce ψ ”), and the *epistemic operators* are \mathbb{K}_a (“agent a knows that”), \mathbb{C}_A (“it is common-knowledge amongst A that”), and \mathbb{D}_A (“the agents in A distributively know that”).

Semantics. Fix an iCGS S . We simultaneously define, by induction on the formulas, $(S, h) \models \varphi$ where $h \in \text{hist}(S)$ and φ is a history formula, and $(S, \pi, m) \models \psi$ where $\pi \in \text{comp}(S)$, $m \geq 0$, and ψ is a path formula:

$$\begin{aligned}(S, h) \models p &\quad \text{iff } \text{last}(h) \in \lambda(p), \text{ for } p \in AP. \\ (S, h) \models \neg\varphi &\quad \text{iff } (S, h) \not\models \varphi. \\ (S, h) \models \varphi_1 \wedge \varphi_2 &\quad \text{iff } (S, h) \models \varphi_i \text{ for } i \in \{1, 2\}. \\ (S, h) \models \langle\langle A \rangle\rangle \psi &\quad \text{iff for some joint strategy } \sigma_A \in \Sigma(S), \\ &\quad \quad (S, \pi, |h|-1) \models \psi \text{ for all } \pi \in \text{out}(h, \sigma_A). \\ (S, h) \models \mathbb{K}_a \varphi &\quad \text{iff for every history } h' \in \text{hist}(S), \\ &\quad \quad h' \equiv_a h \text{ implies } (S, h') \models \varphi. \\ (S, h) \models \mathbb{C}_A \varphi &\quad \text{iff for every history } h' \in \text{hist}(S), \\ &\quad \quad h' \equiv_A^C h \text{ implies } (S, h') \models \varphi. \\ (S, h) \models \mathbb{D}_A \varphi &\quad \text{iff for every history } h' \in \text{hist}(S), \\ &\quad \quad h' \equiv_A^D h \text{ implies } (S, h') \models \varphi.\end{aligned}$$

$$\begin{aligned}(S, \pi, m) \models \varphi &\quad \text{iff } (S, \pi_{\leq m}) \models \varphi, \text{ for } \varphi \text{ a history formula.} \\ (S, \pi, m) \models \neg\psi &\quad \text{iff } (S, \pi, m) \not\models \psi. \\ (S, \pi, m) \models \psi_1 \wedge \psi_2 &\quad \text{iff } (S, \pi, m) \models \psi_i \text{ for } i \in \{1, 2\}. \\ (S, \pi, m) \models \mathbf{X} \psi &\quad \text{iff } (S, \pi, m+1) \models \psi. \\ (S, \pi, m) \models \psi_1 \mathbf{U} \psi_2 &\quad \text{iff for some } j \geq m, (S, \pi, j) \models \psi_2, \text{ and} \\ &\quad \quad \text{for all } k \text{ with } m \leq k < j, \text{ we have} \\ &\quad \quad (S, \pi, k) \models \psi_1.\end{aligned}$$

For a history formula φ , write $S \models \varphi$ to mean that $(S, s) \models \varphi$ for every $s \in S_0$.

We isolate some important fragments.

1. The fragment $\text{ATL}_{K,C,D}$ consists of history formulas φ defined by the grammar above, except with the following path formulas: $\psi ::= \mathbf{X} \varphi \mid \varphi \mathbf{U} \varphi$
2. The fragment ATL (resp. ATL^*) consists of formulas of $\text{ATL}_{K,C,D}$ (resp. $\text{ATL}_{K,C,D}^*$) that do not mention epistemic operators.
3. The CTL operator \mathbf{E} (resp. \mathbf{A}) is definable in ATL^* by $[\![\emptyset]\!]$ (resp. $\langle\langle \emptyset \rangle\rangle$). In particular, $\text{CTL}_{K,C,D}^*$ is a syntactic fragment of $\text{ATL}_{K,C,D}^*$. The fragment of $\text{CTL}_{K,C,D}^*$ consisting of formulas of the form $\mathbf{A} \psi$, where ψ is a path formula, is denoted $\text{LTL}_{K,C,D}$. Finally, LTL is the fragment of $\text{LTL}_{K,C,D}$ that does not mention epistemic operators.

REMARK 1. The definition of the semantics of $\langle\langle A \rangle\rangle \psi$ is the “objective” semantics of $\langle\langle A \rangle\rangle$, and captures the idea that a designer is reasoning about the existence of strategies. On the other hand, “subjective” semantics capture the idea that agents themselves are reasoning about the existence of strategies [31]. In Section 3.1 we define subjective semantics and extend our main result to deal with these.

Model Checking. We state the main decision problem of this work.

DEFINITION 4 (MODEL CHECKING). Let \mathcal{C} be a class of iCGS and \mathcal{F} a sublanguage of $\text{ATL}_{K,C,D}^*$. Model checking \mathcal{C} against \mathcal{F} specifications is the following decision problem: given $S \in \mathcal{C}$ and $\varphi \in \mathcal{F}$ as input, decide whether $S \models \varphi$.

Model checking is undecidable in general. Actually, it is undecidable even if \mathcal{C} consists of all iCGS with $|Ag| = 3$ and \mathcal{F} consists of the single formula $\langle\langle \{1, 2\} \rangle\rangle G p$, see [6]. In Section 3 we prove that model checking PA-iCGS against $\text{ATL}_{K,C,D}^*$ specifications is decidable.

2.3 Examples

We illustrate this definition with two scenarios: the epistemic puzzle of the muddy children [8], and a generalisation of the game Battleships to multiple players. We model (or sketch) the scenario as an iCGS only having public actions, and supply representative formulas of $\text{ATL}_{K,C,D}^*$.

Muddy Children. We express this classic puzzle, as presented, e.g., in [8], in slightly different terms. There are n children, k of them with mud on their foreheads. Each child can see the forehead (and thus the muddy state) of all the other children, but not their own. Then the father enters the scene. The father can see the foreheads of all the muddy children. Each person can only make truthful statements about what they know.

The classic question is: what one statement can the father make that will lead each muddy child to learn that she

is indeed muddy. The answer is that the father declares, assuming that $k \geq 1$, that “at least one of you is muddy”.

There are a number of modeling decisions one can make to formalise this scenario, e.g., we must decide on the exact set of actions. Let $Ag = \{F\} \cup \{1, 2, \dots, n\}$. We give the father two actions, i.e., $Act_F = \{\exists m, \neg \exists m\}$. We give the i th child two actions, i.e., $Act_i = \{\text{know_muddy}, \neg \text{know_muddy}\}$. The set of states is $S = \{0, 1\}^n \times (\text{ACT} \cup \{\epsilon\})$. If the current state is (v, J) then $v_i = 1$ (resp. $v_i = 0$) means that the i th child is (resp. is not) muddy, and J is the most recent joint action (ϵ means that no joint action has yet taken place). The set of initial states is $S_0 = \{0, 1\}^n \times \{\epsilon\}$. The transition relation simply updates the last joint action: $\delta((v, J), J') = (v, J')$. The father is perfectly informed so $\sim_F = \{(s, s) : s \in S\}$, and each child only does not see herself, so $\sim_i = \{((v, J), (v', J')) : \bigwedge_{j \neq i} v_j = v'_j, J = J'\}$. Finally, the atoms are $AP = \{m_i : i \leq n\} \cup \{\alpha_a : \alpha \in Act_a, a \in Ag\}$. Finally, define $\lambda(m_i) = \{(s, J) : s_i = 1\}$ and $\lambda(\alpha_a) = \{(s, J) : J(a) = \alpha\}$. So m_i means that the i th child is muddy, and α_a means that agent a 's last action was $\alpha \in Act_a$. It is straightforward to check that we have defined an iCGS only having public actions.

Define the shorthand formula $\text{Kw}_a \phi$, read “agent a knows whether ϕ ”, by the formula $\text{K}_a \phi \vee \text{K}_a \neg \phi$. Consider the formula:

$$\langle\langle \{f, 1, \dots, n\} \rangle\rangle [(\bigwedge_{i \leq n} G(X \alpha_a \rightarrow \widehat{\alpha_a})) \wedge F(\bigwedge_{i \leq n} \text{Kw}_i m_i)]$$

where the first conjunction is over $a \in Ag, \alpha \in Act_a$, and $\widehat{\alpha_a}$ is a formula representing the intended meaning of α_a , e.g., if $\alpha_a = \text{know_muddy}$ then $\widehat{\alpha_a} = \text{Kw}_a m_a$. This expresses that all the agents have a truthful strategy so that eventually each child will know whether or not she is muddy.

The reader might wonder what would happen if we don't explicitly express that the actions are truthful. Consider the following formula that says that the father has a strategy such that eventually the children know their muddy state:

$$\langle\langle \{f\} \rangle\rangle F \bigwedge_{i \leq n} \text{Kw}_i m_i$$

This formula is true. Indeed, a strategy for the father is to play action $\exists m$ at the i th step iff the i th child is muddy.

In the next scenario, what agents see changes over time (unlike in the muddy children scenario). In both scenarios, what agents hear gets updated over time.

Battleships. We consider a game of battleships with three players. Each player has a 10×10 -board with numeric coordinates from bottom-left $(0, 0)$ to top-right $(9, 9)$. Initially, each player can only see her own board. On her board each player displays her battleships: one carrier of size 5, two battleships of size 4, three cruisers of size 3, four submarines also of size 3, and five destroyers of size 2. We assume that ships are displayed either horizontally or vertically. As is standard, overlapping is not allowed.

Here we consider a synchronous version of battleships, structured in 2-phase rounds, where in phase 1 every player broadcasts the name p of a player and cell (n, m) of p 's board to attack. Then, in phase 2, the players truthfully state whether they have been hit or missed and the boards are updated accordingly. A player is eliminated once all her ships have been destroyed. The last player standing, if there is one, wins the game. The relevant atoms in this game are

win_i and $lose_i$ which state whether player i has won or lost. It is a routine exercise to build an iCGS only having public actions from this description. We only mention that the assumption that players are truthful can be built in to the iCGS, i.e., by limiting the available actions a player has in state 2.

Now consider the formula:

$$\langle\langle \{1, 2\} \rangle\rangle F \langle\langle \{1, 3\} \rangle\rangle F (lose_2 \vee \langle\langle \{1\} \rangle\rangle F win_1)$$

This expresses that player 1 can collude with each of her enemies, in order to weaken player 2 or win the game.

3. DECIDABILITY OF PA-iCGS

In this section we prove that model checking PA-iCGS against $\text{ATL}_{\mathbb{K}, \mathbb{C}, \mathbb{D}}^*$ specifications is decidable. This should be contrast with the fact that model checking arbitrary iCGS against ATL specifications is undecidable [6].

THEOREM 1. *Model checking PA-iCGS against $\text{ATL}_{\mathbb{K}, \mathbb{C}, \mathbb{D}}^*$ specifications is 2EXPTIME-complete.*

The bulk of this section is devoted to proving decidability. We then establish the complexity, and discuss further extensions of the result. Before giving the proof, we introduce an encoding μ of histories.

DEFINITION 5. *Let \mathbf{S} be an iCGS. Let $\mu : S_0 \times \text{ACT}^* \rightarrow \text{hist}(\mathbf{S})$ denote the function mapping (s_0, u) to the history starting at the initial state s_0 that results from the sequence of joint actions $u \in \text{ACT}^*$. That is, $\mu(s_0, u)$ is the history h such that $h_0 = s_0$, $h_j = \delta(h_{j-1}, u_{j-1})$ for $1 \leq j \leq |u|$.*

For PA-iCGS, the encoding is actually a bijection:

REMARK 2. *Let \mathbf{S} be a PA-iCGS. Since each \sim_a is reflexive, $\delta(s, \cdot) : \text{ACT} \rightarrow S$ is injective for every $s \in S$. Thus, $\mu : S_0 \times \text{ACT}^* \rightarrow \text{hist}(\mathbf{S})$ is a bijection. In particular, for every $h \in \text{hist}(\mathbf{S})$ and $s \in S_0$ there exists a unique $u \in \text{ACT}^*$ such that $\mu(s, u) = h$. This bijection allows us to encode histories of \mathbf{S} by (unique) elements of $S_0 \times \text{ACT}^*$.*

An immediate consequence of having only public actions, but one that forms the foundation of our decidability proof, is that the moment different joint actions are taken, two histories become distinguishable.

LEMMA 1. *Let \mathbf{S} be a PA-iCGS. For all $a \in Ag$, $u, u' \in \text{ACT}^*$ and $s, s' \in S_0$, if $\mu(s, u) \equiv_a \mu(s', u')$ then $u = u'$.*

PROOF. If $\mu(s, u) \equiv_a \mu(s', u')$ then $|u| = |u'|$ and, for all $0 \leq j \leq |u|$, $\mu(s, u)_j \sim_a \mu(s', u')_j$. By the definition of having only public actions, $u_j = u'_j$ for all $j < |u|$. \square

We prove Theorem 1 in the rest of this section. We use an automata-based marking algorithm. Such algorithms have been successfully applied to a number of logics, including CTL^* [19] and ATL^* [1] in the perfect information setting.

Automata theory. Since our proof uses an automata-theoretic approach we now fix notations of word and tree automata. We remark that we only make use of standard properties of automata operating on finite words, infinite words, and infinite trees [20].

A *deterministic finite-word automaton* (DFW) is a tuple $M = (\Sigma, S, s_0, \rho, F)$ where Σ is the *input alphabet*, S is the finite set of *states*, $s_0 \in S$ the *initial state*, $\rho : S \times \Sigma \rightarrow S$

the deterministic transition function, and $F \subseteq S$ the set of final states. The run of M on $u \in \Sigma^*$ is the finite sequence $s_0s_1 \dots s_{|u|}$ where $\rho(s_i, u_i) = s_{i+1}$ for all $i < |u|$. The automaton accepts a word $u \in \Sigma^*$ iff the run of M on u ends in a final state. A DFW is empty if it accepts no word. A set of strings $X \subseteq \Sigma^*$ is called regular if there is a DFW M that accepts $u \in \Sigma^*$ iff $u \in X$.

We also make use of automata operating on infinite words $\alpha \in \Sigma^\omega$: a deterministic parity word automata (DPW) is a tuple $P = (\Sigma, S, s_0, \rho, c)$ where all components are as for DFW except that $c : S \rightarrow \mathbb{Z}$ is the colouring function. The run of P on $\alpha \in \Sigma^\omega$ is the infinite sequence $s_0s_1 \dots$ such that $\rho(s_i, \alpha_i) = s_{i+1}$ for all i . The automaton accepts a word α iff the smallest color k for which there are infinitely many i with $c(s_i) = k$ is even (where $s_0s_1 \dots$ is the run of M on α).

We also make use of automata operating on trees. A deterministic parity tree automata (DPW) is a tuple $T = (\Sigma, D, S, s_0, \rho, c)$ where all components are as for a DPW except that D is the finite set of directions, and $\rho : S \times \Sigma \rightarrow S^D$. The automaton operates on Σ -labelled D -ary branching trees, i.e., functions $f : D^* \rightarrow \Sigma$. A branch of t is an infinite sequence $\beta \in D^\omega$. The run of T in input t is the Q -labeled D -ary branching tree $g : D^* \rightarrow Q$ such that $g(\epsilon) = s_0$ and $g(ud) = \rho(g(u), t(u))$. The automaton accepts the tree f iff for every $\beta \in D^\omega$ (called a branch), the smallest color k for which there are infinitely i with $g(\beta_i) = k$ is even.

The classes of DFW and DPW are effectively closed under the Boolean operations (complementation and intersection). Also, DFW, DPW and DPT can be effectively tested for emptiness. Finally, we make use of the following important fact connecting linear temporal logic with automata:

PROPOSITION 1 ([35, 29]). *Every LTL formula ψ over atoms AP can be effectively converted into a DPW P_ψ with input alphabet 2^{AP} that accepts a word $\alpha \in 2^{AP}$ iff $\alpha \models \psi$. Moreover, the DPW has double-exponentially many states and single-exponentially many colours.*

Proof outline. We proceed by induction on the formula φ to be checked. We build a DFW that accepts all encodings of histories h such that $(S, h) \models \varphi$. Precisely, we build a DFW M_φ^s that accepts a sequence of joint actions $u \in \text{ACT}^*$ iff $(S, \mu(s, u)) \models \varphi$. The atomic case is immediate, and the Boolean cases follow from the effective closure of DFW under complementation and intersection. The $\varphi = \mathbb{K}_a \varphi'$ case is done by simulating the DFW $M_{\varphi'}^t$ for $t \in S_0$, and recording whether or not $\mu(s, u) \sim_a \mu(t, u)$; the other knowledge operators are similar. The strategic operator $\varphi = \langle\!\langle A \rangle\!\rangle \psi$ is done as follows: first we show that we can assume ψ is an LTL formula, and then we build a DPW for the formula ψ ; we build the DFW that simulates the DPW, and when the input ends we use a tree automaton to decide if there is a joint strategy that ensures that the DPW accepts all computations consistent with that joint strategy.

Generalisation of the labeling function. We first generalise the labeling function of iCGS so that atoms are regular sets of histories (instead of state labelings).

DEFINITION 6. A generalised iCGS is a tuple

$$S = \langle Ag, AP, \{Act_a\}_{a \in Ag}, S, S_0, \delta, \{\sim_a\}_{a \in Ag}, \Lambda \rangle$$

where all entries are as for iCGS, except that $\lambda : AP \rightarrow 2^S$ is replaced by a function $\Lambda : AP \rightarrow 2^{\text{hist}(S)}$ such that $\Lambda(p) \subseteq$

$\text{hist}(S)$ is a regular set of histories, i.e., there exists a DFW over the alphabet S accepting $h \in \text{hist}(S)$ iff $h \in \Lambda(p)$.

Then, we redefine the atomic case of the semantics of $\text{ATL}_{\mathbb{K}, \mathbb{C}, \mathbb{D}}^*$: $(S, h) \models p$ iff $h \in \Lambda(p)$. It is immediate that generalised iCGS are indeed more general than iCGS:

LEMMA 2. Let S be an iCGS with labeling function λ . The generalised iCGS S' with labeling $\Lambda(p) = \{h \in \text{hist}(S) : \text{last}(h) \in \lambda(p)\}$ has the property that $S \models \varphi$ iff $S' \models \varphi$ (for all formulas φ of $\text{ATL}_{\mathbb{K}, \mathbb{C}, \mathbb{D}}^*$).

PROOF. First, note that $\Lambda(p)$ is regular since a DFW can read the history h and store in its state whether or not the last state it read is in $\lambda(p)$ or not. Second, the fact that $S \models \varphi$ iff $S' \models \varphi$ holds by a straightforward induction on the structure of φ . \square

A generalised PA-iCGS is a generalised iCGS that only has public actions, i.e., it satisfies the condition in Definition 3 (which does not depend on the labeling). For the rest of the proof we view S as a generalised PA-iCGS.

Inductive Statement. Let S be a generalised PA-iCGS. For every history formula φ and initial state $s \in S_0$ we will build a DFW M_φ^s such that for every $u \in \text{ACT}^*$,

$$M_\varphi^s \text{ accepts } u \text{ iff } (S, \mu(s, u)) \models \varphi.$$

From this it is easy to get the decidability stated in the theorem: simply check that ϵ is accepted by every M_φ^s with $s \in S_0$.

We build the DFW M_φ^s , simultaneously for all $s \in S_0$, by induction on φ .

φ is an atom. Say $\varphi = p \in AP$ and let $s \in S_0$. The required DFW M_φ^s should accept $u \in \text{ACT}^*$ iff $\mu(s, u)$ is accepted by the DFW $\Lambda(p)$. To do this we define M_φ^s to simulate S and the DFW $R = (S, Q, q_0, \rho, F)$ for $\Lambda(p)$ in parallel, i.e., by taking a product of S and R . Formally, define $M_\varphi^s = (\text{ACT}, S \times Q, (\iota, q_0), \tau, F')$ where the transition function τ maps state (s, q) and input $a \in \text{ACT}$ to state $(\delta(s, a), \rho(q, s))$, and the final states F' are of the form (s, q) where $\rho(q, s) \in F$.

φ is a Boolean combination. Let $s \in S_0$. The Boolean combinations follow from the effective closure of DFW under complementation and intersection. Indeed, $M_{\neg\varphi}^s$ is formed by complementing the final states of M_φ^s , and $M_{\varphi_1 \wedge \varphi_2}^s$ is the product of the $M_{\varphi_i}^s$'s.

φ is of the form $\mathbb{K}_a \varphi'$. Let $s \in S_0$. By induction, we have DFW $M_{\varphi'}^t$ for $t \in S_0$. The required DFW should accept a string u iff, for every $t \in S_0$, if $\mu(s, u) \equiv_a \mu(t, u)$ then $M_{\varphi'}^t$ accepts u .

To do this, the DFW will simulate, in parallel, each $M_{\varphi'}^t$ for $t \in S_0$. This is done by forming their product, i.e., the states of the product are $q : S_0 \rightarrow Q$ where Q is the union of the state sets of the $M_{\varphi'}^t$ for $t \in S_0$, and there is a transition in the product from q to q' on input $J \in \text{ACT}$ if for each $t \in S_0$ there is a transition in $M_{\varphi'}^t$ from $q(t)$ to $q'(t)$ on input J . Instrument this product by recording, on input $u \in \text{ACT}^*$ a function $f_u : S_0 \rightarrow S$ and a set $G_u \subseteq S_0$ with the following properties:

- $f_u(t) = \text{last}(\mu(t, u))$
- $t \in G_u$ iff for every prefix v of u , $f_v(s) \sim_a f_v(t)$ (thus, initially $G_\epsilon = \{t \in S_0 : t \sim_a s\}$, and the moment $f_v(s) \not\sim_a f_v(t)$ we remove t from G_v).

A state $\langle f, G, q \rangle$ is final if, for every $t \in G$ it is also the case that $q(t)$ is a final state of M_φ^t .

φ is of the form $\mathbb{C}_A\varphi'$. This is identical to the \mathbb{K}_a case except replace \sim_a by $\sim_A^\mathbb{C}$ and replace \equiv_a by $\equiv_A^\mathbb{C}$.

φ is of the form $\mathbb{D}_A\varphi'$. This is identical to the \mathbb{K}_a case except replace \sim_a by $\sim_A^\mathbb{D}$ and replace \equiv_a by $\equiv_A^\mathbb{D}$.

φ is of the form $\langle\langle A \rangle\rangle\psi$. We proceed in two steps. First, we show how to linearise path formulas, and then we show how to encode strategies as trees so that we can build the promised DFW.

Linearising path formulas. One can think of an $\text{ATL}_{\mathbb{K},\mathbb{C},\mathbb{D}}^*$ path formula ψ as an LTL formula $\text{lin}(\psi)$ over a fresh set of atoms $\text{max}(\psi)$, the maximal history subformulas of ψ . This translation of $\text{ATL}_{\mathbb{K},\mathbb{C},\mathbb{D}}^*$ path formulas to LTL formulas does not make use of the assumption that the iCGS S only has public actions, and it is analogous to the translation of ATL^* (or CTL^*) path formulas to LTL formulas over maximal state subformulas [19, 1].

We briefly discuss the translation. Let $\text{max}(\psi)$ be the set of history subformulas of ψ that are maximal, i.e., a history formula $\varphi \in \text{max}(\psi)$ iff it occurs in ψ and that occurrence is not a subformula of any other occurrence of a history subformula of ψ . If ψ is a path formula, define $\text{lin}(\psi)$ to be the LTL formula where for each $\varphi \in \text{max}(\psi)$ there is a fresh atom $\boxed{\varphi}$ such that every occurrence of φ in ψ is replaced by $\boxed{\varphi}$. For example, consider $\psi = (p \mathbb{U} \langle\langle A \rangle\rangle \mathbb{K}_a p) \vee \mathbb{X} \neg \mathbb{C}_A p$. The history subformulas of ψ are $\{p, \langle\langle A \rangle\rangle \mathbb{K}_a p, \mathbb{K}_a p, \neg \mathbb{C}_A p, \mathbb{C}_A p\}$. Then $\text{max}(\psi) = \{p, \langle\langle A \rangle\rangle \mathbb{K}_a p, \neg \mathbb{C}_A p\}$.¹ Thus $\text{lin}(\psi)$ is the LTL formula $(\boxed{p} \mathbb{U} \boxed{\langle\langle A \rangle\rangle \mathbb{K}_a p}) \vee \mathbb{X} \boxed{\neg \mathbb{C}_A p}$ over the atoms $\boxed{\varphi}$ for $\varphi \in \text{max}(\psi)$. Since, by induction, we have built DFW for each boxed atom, for the remainder of the proof we assume that ψ is an LTL formula.

Construction of $M_{\langle\langle A \rangle\rangle\psi}^s$ for an LTL formula ψ . We will build a DPW $E_{\psi,s}$ over ACT that accepts $\alpha \in \text{ACT}^\omega$ iff $(S, \mu(s, \alpha)) \models \psi$. The promised DFW $M_{\langle\langle A \rangle\rangle\psi}^s$ reads $u \in \text{ACT}^*$ and simulates $E_{\psi,s}$. Suppose after reading u the state of $E_{\psi,s}$ is q . Then $M_{\langle\langle A \rangle\rangle\psi}^s$ accepts, i.e., q is defined to be a final state of $M_{\langle\langle A \rangle\rangle\psi}^s$, iff there exists uniform σ_A such that for every $\alpha \in \text{ACT}^\omega$, if $\mu(s, u \cdot \alpha) \in \text{out}(S, \mu(s, u), \sigma_A)$ then α is accepted by $E_{\psi,s}$ starting from state q (and thus $(S, \mu(s, u)) \models \langle\langle A \rangle\rangle\psi$, as required). These latter realisability problems (one for each q) are solved offline by constructing DPT $F_{\psi,s,q}$ and testing them for non-emptiness. We now show how to build the automata $E_{\psi,s}$ and $F_{\psi,s,q}$.

Construction of DPW $E_{\psi,s}$. To build $E_{\psi,s}$, begin by writing $AP(\psi)$ for the finite set of atoms appearing in ψ . First, for each $p \in AP(\psi)$, let D^p be a DFW for the regular set $\{u \in \text{ACT}^* : \mu(s, u) \in \Lambda(p)\}$. Second, by Proposition 1, every LTL formula ψ can be converted into a DPW D_ψ over alphabet 2^{AP} that accepts all word models of that formula. Let Q_ψ be the states and $\Delta_\psi : Q_\psi \times 2^{AP(\psi)} \rightarrow Q_\psi$ the transition of the DPW. Now, the DPW $E_{\psi,s}$ simulates D_ψ and each DFW D^p . The automaton does this by storing and updating a state $q_u \in Q_\psi$ and a function f_u such that $f_u(p)$ is the state of D^p (i.e., $f_u : AP(\psi) \rightarrow Q$ where Q is the union of states of the D^p s). Transitions of $E_{\psi,s}$ are as follows: from state $\langle q_u, f_u \rangle$ and input $d \in \text{ACT}$ the next state $\langle q_{ud}, f_{ud} \rangle$ satisfies that $q_{ud} = \Delta_\psi(q_u, Z)$ where $p \in Z$ iff $f_u(p)$ is a final state of D^p , and $f_{ud}(p)$ is the state resulting from applying

¹Note that although p has a non-maximal occurrence in ψ , it is included in $\text{max}(\psi)$ since it has at least one occurrence which is maximal, i.e., on the left-side of U.

the transition function of D^p to the state $f_u(p)$ and input d . Define the colour of $\langle q, f \rangle$ to be the same as the colour in D_ψ of the state q , and $\langle q, f \rangle$ is an initial state if q is an initial state in D_ψ . The following is straightforward to prove:

LEMMA 3. *The DPW $E_{\psi,s}$ accepts $\alpha \in \text{ACT}^\omega$ if and only if $(S, \mu(s, \alpha)) \models \psi$.*

Construction of DPT $F_{\psi,s,q}$. To solve the synthesis problem we will encode strategies as trees and use tree-automata. Encode a strategy σ of agent a by the ACT-branching tree

$$T_\sigma : \text{ACT}^* \rightarrow (\text{Act}_a)^{S_0}$$

where, for $u \in \text{ACT}^*$, $T_\sigma(u)(t) = \sigma(\mu(t, u))$. By Lemma 1, σ is uniform iff T_σ satisfies the property

$$\mu(t, u) \equiv_a \mu(t', u) \Rightarrow T_\sigma(u)(t) = T_\sigma(u)(t') \quad (1)$$

LEMMA 4. *The set of encodings T_σ of uniform strategies σ of agent a is recognised by a DPT.*

PROOF. To do this, it is sufficient to build a DPT that accepts a tree T iff T has property (1). Informally, for every pair of different states $t, t' \in S_0$, the DPT keeps a bit that is initialised to 1 (signifying that it must verify that $T(u)(t) = T(u)(t')$), and as soon as it finds that $\mu(t, u) \not\equiv_a \mu(t', u)$ it sets the bit to 0, which signals that it no longer has to ensure $T_\sigma(u)(t) = T_\sigma(u)(t')$. \square

Encode a set of uniform strategies σ_A (for $A \subseteq Ag$) as the convolution T_{σ_A} of their individual encodings, i.e.,

$$T_{\sigma_A} : \text{ACT}^* \rightarrow \prod_{a \in A} (\text{Act}_a)^{S_0}$$

where $T_{\sigma_A}(u)(a) = T_{\sigma_a}(u)$. Running the automata for σ_a in parallel yields:

LEMMA 5. *For every $A \subseteq Ag$, the set of encodings of joint uniform strategies σ_A is recognised by a DPT.*

Finally, in order to solve the synthesis problem for state q , we define a DPT $F_{\psi,s,q}$ that accepts T_{σ_A} iff for every $\alpha \in \text{ACT}^\omega$, if $\mu(s, \alpha)$ is consistent with σ_A then α is accepted by $E_{\psi,s}$ starting from q .

The DPT $F_{\psi,s,q}$ works as follows: it reads T_{σ_A} as input and, on every branch $\alpha \in \text{ACT}^\omega$ of the tree, simulates $E_{\psi,s}$ starting from q , and accepts that branch if both $E_{\psi,s}$ is accepting and $\mu(s, \alpha)$ is consistent with σ_A . This can be done by a tree-automaton because a) $E_{\psi,s}$ is deterministic and thus it can be run on all paths of the tree (i.e., this step would not be possible if $E_{\psi,s}$ were a non-deterministic automaton); and b) checking if $\mu(s, \alpha)$ is consistent with σ_A is simply a matter of checking that $\alpha_0(a) = \sigma_A(a)(s)$ and $\alpha_{i+1}(a) = \sigma_A(a)(\mu(s, \alpha_{\leq i}))$, for every $i \in \mathbb{N}$ and $a \in A$.

Final states of DFW $M_{\langle\langle A \rangle\rangle\psi}^s$. Finally, define the state q of $M_{\langle\langle A \rangle\rangle\psi}^s$ to be a final state iff the DPT $F_{\psi,s,q}$ is non-empty (a decidable condition).

This completes the construction of $M_{\langle\langle A \rangle\rangle\varphi}^s$, and the proof of decidability.

3.1 Complexity

In this section we analyse the complexity of our decision procedure for a fixed number of agents. We then establish the lower bound by a reduction from a problem known to be 2EXPTIME-hard.

First, we calculate $\|\varphi\|$, the number of states of M_φ^s , for each case:

1. Atomic: $\|p\| = O(1)$ for $p \in AP$.
2. Negation: $\|\neg\varphi\| = \|\varphi\|$.
3. Conjunction: $\|\varphi \wedge \varphi'\| = \|\varphi\| \times \|\varphi'\|$.
4. Epistemic: $\|\mathbb{K}_a \varphi\| = (2 \times |S| \times \|\varphi\|)^{|S|}$, and the same for $\|\mathbb{C}_A \varphi\|$ and $\|\mathbb{D}_A \varphi\|$.
5. Strategic: $\|\langle\langle A\rangle\rangle\psi\| = 2^{2^{O(|lin(\psi)|)}} \times \|\tilde{\psi}\||^{AP(lin(\psi))} \times O(2^{|S|^2})$ where $lin(\psi)$ is the linearisation of ψ , $\tilde{\psi}$ is the largest history subformula of ψ , and $AP(\cdot)$ is the set of atomic predicates occurring in its argument.

The last case requires some explanation. The DPT accepting the set of all joint-strategies σ_A has size $O(2^{|S|^2})$, and has two colours. The DPW D_ψ has double-exponentially many states and single-exponentially many colours (in the size of $lin(\psi)$) [35, 29]. The DPW $E_{\psi,s}$ has $2^{2^{O(|lin(\psi)|)}} \times \|\tilde{\psi}\||^{AP(\psi)}$ many states. The DPT $F_{\psi,s,q}$ has $2^{2^{O(|lin(\psi)|)}} \times \|\tilde{\psi}\||^{AP(\psi)} \times O(2^{|S|^2})$ many states and $2^{O(|lin(\psi)|)}$ many colours. This gives the stated value of $\|\langle\langle A\rangle\rangle\psi\|$.

Second, we calculate the time for constructing the DFW M_φ^s . In the first four cases this cost is polynomial in the size of the DFW, i.e., $\|\varphi\|$. For the strategy case we incur a cost to calculate the final states, i.e., solving the emptiness of the DPT $F_{\psi,s,q}$. The cost of solving the emptiness of a DPT with n states and m colours is at most $n^{O(m)}$ [16]. Thus, the time for constructing M_φ^s is at most $n^{O(m)}$ where $n = 2^{2^{O(|lin(\psi)|)}} \times \|\tilde{\psi}\||^{AP(lin(\psi))} \times O(2^{|S|^2})$ and $m = 2^{O(|lin(\psi)|)}$.

Finally, let $z = |S| + |\varphi|$. The time for constructing M_φ^s of each step of the procedure can be bounded above by $2^{2^{O(z)}} \times |\Lambda|^{2^{O(z)}}$ where $|\Lambda|$ is the size of the largest DFW representing atoms of the generalised PA-iCGS (a maximum exists since AP is finite). Since there are at most z such steps, the time for constructing, and thus the size, of the resulting DFW is $2^{2^{O(z^2)}}$. Testing if ϵ is accepted by this automaton has no additional cost. Thus, our algorithm runs in 2EXPTIME.

Lower-Bound. To prove the lower bound in Theorem 1 we reduce from a known 2EXPTIME-hard problem, i.e., model checking a CGS with $Ag = \{a, b\}$ against a formula of the form $\langle\langle \{a\} \rangle\rangle\psi$ for an LTL formula ψ [27, 28]. One can translate a two-player CGS S into a polynomially larger CGS S' with public actions such that $S \models \varphi$ iff $S' \models \varphi$. Indeed, the agents, actions, and atoms are the same, $S' = S \times (\text{ACT} \cup \{\epsilon\})$, $S'_0 = S_0 \times \{\epsilon\}$, $\delta'((s, d), d') = (\delta(s, d'), d')$, and $\lambda'(p) = \{(s, d) : s \in \lambda(p)\}$ (note that because S has two-players, $|\text{ACT}| = |\text{Act}_1| \times |\text{Act}_2|$, and thus $|S'|$ is polynomial in the size of S).

3.2 Subjective Semantics

In this section we show that our result still holds if we use subjective semantics instead of objective semantics.

Our definition of semantics of $\langle\langle A\rangle\rangle\psi$ is called “objective” (see Remark 1). An alternative definition is called “subjective”: replace “for all $\pi \in \text{out}(h, \sigma_A)$ ” by “for all $\pi \in \bigcup_{a \in A, h' \equiv_a h} \text{out}(S, h', \sigma_A)$ ” in the semantics $(S, h) \models \langle\langle A\rangle\rangle\psi$. Subjective semantics expresses, intuitively, that the agents A know that a given strategy will guarantee a certain outcome. We remark that in this case \mathbb{K}_a is definable in terms of $\langle\langle A\rangle\rangle$, i.e., $\langle\langle a \rangle\rangle\varphi \mathbf{U} \varphi$.

The decidability proof of Theorem 1 is for objective semantics. To deal with subjective semantics proceed as follows. For $u \in \text{ACT}^*$ let $T_u^s \subseteq S_0$ be the set of t such that there exists $a \in A$ with $t \equiv_a s$. The DFW $M_{\langle\langle A\rangle\rangle\psi}^s$ simulates $E_{\psi,t}$ for all $t \in T_u^s$. After reading u , each automaton $E_{\psi,t}$ for $t \in T_u^s$ is in some state, say q_t . The DFW is required to accept u iff there exists a joint strategy σ_A such that for every $\alpha \in \text{ACT}^*$ and $t \in T_u^s$, if $\mu(t, u \cdot \alpha) \in \text{out}(S, \mu(t, u), \sigma_A)$ then α is accepted by $E_{\psi,t}$ starting from q_t . In order to decide the right-hand side we build the DPT F_{ψ,t,q_t} for $t \in T_u^s$ (as we did above for s). Then we check if the intersection of the automata F_{ψ,t,q_t} (for $t \in T_u^s$) is non-empty.

4 COMPARISON WITH BROADCAST ENVIRONMENTS

The work most closely related to ours is [33] in which the authors show that one can decide if a given formula of knowledge and linear-time, which we denote LK , is realisable (by a tuple of uniform strategies) assuming that the environment is a “broadcast environment”. In this section we denote the logic from [33] by LK .

The relationship with [33] is twofold: i) the realisability problem for LK can be reduced to model checking $\text{ATL}_\mathbb{K}^*$ specifications, and ii) our notion of “having only public actions” (Definition 3) is orthogonal to “broadcast environments”. We now supply the justifications for i) and ii).

i) LK realisability can be reduced to model-checking $\text{ATL}_\mathbb{K}^*$. We show how to reduce the realisability problem for LK to the model checking problem for $\text{ATL}_\mathbb{K}^*$. To do so, we first recall the syntax and semantics of LK from [33]. The syntax is defined as the set of formulas ψ generated by the following grammar:

$$\psi ::= p \mid \neg\psi \mid \psi \wedge \psi \mid \mathbf{X} \psi \mid \psi \mathbf{U} \psi \mid \mathbb{K}_a \psi$$

where $p \in AP$, $a \in Ag$, and $A \subseteq Ag$ is non-empty.

The semantics \models_{LK} is defined over (S, π, m) where S is an iCGS, $\pi \in \text{comp}(S)$ and $n \in \mathbb{N}$. We denote the satisfaction relation \models_{LK} to distinguish it from \models . The Boolean and temporal operators are as usual:

$$\begin{aligned} (S, \pi, m) \models_{\text{LK}} p &\quad \text{iff } \pi_m \in \lambda(p) \\ (S, \pi, m) \models_{\text{LK}} \neg\psi &\quad \text{iff } (S, \pi, m) \not\models_{\text{LK}} \psi \\ (S, \pi, m) \models_{\text{LK}} \psi_1 \wedge \psi_2 &\quad \text{iff } (S, \pi, m) \models_{\text{LK}} \psi_i \text{ for } i \in \{1, 2\} \\ (S, \pi, m) \models_{\text{LK}} \mathbf{X} \psi &\quad \text{iff } (S, \pi, m+1) \models_{\text{LK}} \psi \\ (S, \pi, m) \models_{\text{LK}} \psi_1 \mathbf{U} \psi_2 &\quad \text{iff for some } j \geq m, (S, \pi, j) \models_{\text{LK}} \psi_2, \\ &\quad \text{and for all } k \text{ with } m \leq k < j, \\ &\quad (S, \pi, k) \models_{\text{LK}} \psi_1 \end{aligned}$$

The epistemic operator \mathbb{K}_a is follows:

$$(S, \pi, m) \models_{\text{LK}} \mathbb{K}_a \psi \text{ iff } (S, \pi', m') \models_{\text{LK}} \psi \text{ for all } \pi' \in \text{comp}(S) \text{ such that } \pi_{\leq m} \equiv_a \pi'_{\leq m'} \text{ (in particular, } m = m').$$

An LK -formula ψ is *realisable* if there exists a uniform strategy σ_A such that for all $s_0 \in S_0$ and all $\pi \in \text{out}(S, s_0)$, we have that $(S, \pi, 0) \models \psi$.

We now present the reduction. Let ψ be a formula of LK . Define $\hat{\psi}$, a path formula of $\text{CTL}_\mathbb{K}^*$, by recursively replacing $\mathbb{K}_a \psi$ by $\mathbb{K}_a \mathbf{A} \hat{\psi}$. We claim that for all iCGS S , ψ is realisable in S iff $S \models \langle\langle Ag \rangle\rangle \hat{\psi}$. To see this it is sufficient to establish the following inductive hypothesis: $(S, \pi, m) \models_{\text{LK}} \psi$ iff $(S, \pi, m) \models \hat{\psi}$. To prove the inductive hypothesis use that the following are equivalent to $(S, \pi, m) \models_{\text{LK}} \mathbb{K}_a \psi$:

1. $(S, \pi', m) \models_{LK} \psi$ for all $\pi' \in \text{comp}(S)$ such that $\pi'_{\leq m} \equiv_a \pi_{\leq m}$ (by the definition of \models_{LK});
2. $(S, \pi', m) \models \hat{\psi}$ for all $\pi' \in \text{comp}(S)$ such that $\pi'_{\leq m} \equiv_a \pi_{\leq m}$ (by the inductive hypothesis);
3. $(S, \pi, m) \models \mathbb{K}_a A \hat{\psi}$ (by definition of \models).

ii) Broadcast environments and PA-iCGS are incomparable. We briefly describe how [33] models “broadcast environment”. That work defines an interpreted systems (see [8] for background on these) in which each agent’s local state consists of a private part (that only depends on its local actions) and a shared part (that depends on the joint actions, but that is the same for all agents). In our terminology a broadcast environment is an iCGS with $Ag = \{e, 1, 2, \dots, n\}$ whose state set is of the form $S = L_e \times \prod_{i \leq n} L_i$ (for some finite sets L_a for $a \in Ag$), and whose transition maps state (l_e, l_1, \dots, l_n) and joint-action $J \in ACT$ to the state $(\tau_e(l_e, J), \tau_1(l_1, J(1)), \dots, \tau_n(l_n, J(n)))$ where $\tau_e : L_e \times ACT \rightarrow L_e$ and $\tau_i : L_i \times Act_i \rightarrow L_i$ are functions (similar to the evolution functions in interpreted systems), except that L_i for $i \neq e$ does not depend on joint-actions, only on local actions). Finally, define $(l_e, l_1, \dots, l_n) \sim_i (l'_e, l'_1, \dots, l'_n)$ iff $l_i = l'_i$ and $F(l_e) = F(l'_e)$ where $F : L_e \rightarrow O$ is a function mapping environment states to some fixed set O of observations (note that F and O are independent of i , and thus all agents have the same observation of the environment).

Now, the set of PA-iCGS is incomparable (wrt. subset) with these iCGS. On the one hand, setting $L_e = ACT = O$ and F to be the identity function, results in an iCGS having only public actions. On the other hand, we allow L_i to depend on the joint-actions (not just the local actions).

5. CONCLUSIONS

In this paper we put forward a class of CGS with imperfect information, namely the iCGS only having public actions, which admit a decidable model checking problem, even in the presence of perfect recall. This is in contrast with the fact that even realisability of safety properties on arbitrary iCGS is undecidable [6]. Specifically, we considered a rich formal language to express complex strategic and epistemic properties of agents in MAS. This is the extension $ATL_{K,C,D}^*$ of the alternating-time temporal logic ATL^* , with operators for individual, common, and distributed knowledge. We provided these languages with a semantics in terms of iCGS, according to both the objective and subjective interpretation of ATL modalities. Most importantly, we identified a subclass of iCGS – those having only public actions, or PA-iCGS – for which we were able to prove that the model-checking problem is decidable. The interest of these results lies in the fact that PA-iCGS capture many important MAS scenarios, including certain games of imperfect information, epistemic puzzles, blackboard systems, face to face communication, etc. Indeed, all scenarios mentioned in previous work on broadcast environments [23, 33] can be captured by PA-iCGS.

A number of extensions of ATL^* have been proposed in order to express classic solutions concepts (like Nash Equilibria) [12, 13, 24, 25]. The decidability of model checking PA-iCGS against epistemic extensions of these strategy logics is currently unexplored.

Notwithstanding their generality, there are many features of MAS that are not naturally expressed within PA-iCGS or broadcast environments. We discuss some of them:

Asynchronous recall. Social media like Twitter make use of public actions, but are more naturally modeled as asynchronous MAS (rather than synchronous systems, as we do).

Bounded-recall. Games like Bridge and Stratego are interesting to play in part because humans have to remember some of the history of a play, a feature that might be modeled by bounded recall (rather than perfect recall). However, restricting agents to finite-memory strategies also results in undecidability on arbitrary iCGS [36]. On the other hand, our proof implies that if a formula $\langle\langle A \rangle\rangle \psi$ is true then there are finite-memory strategies witnessing this fact, and if a formula $\mathbb{K}_a \varphi$ is true then there is a finite-state machine that accepts exactly the histories making $\mathbb{K}_a \varphi$ true. This suggests that our results can be used to model agents of bounded-recall.

Probabilities. Several scenarios, such as card games and security protocols, involve probability either at the level of the iCGS or at the level of strategies.

In future work we plan to investigate the points raised above, as well as to develop optimal model checking algorithms for fragments of $ATL_{K,C,D}^*$ and to implement them in an extension of the MCMAS tool for MAS verification [22]. **Acknowledgements.** F. Belardinelli acknowledges the support of the ANR JCJC Project SVeDaS (ANR-16-CE40-0021). S. Rubin is a Marie Curie fellow of the Istituto Nazionale di Alta Matematica. The authors thank Benjamin Aminof for fruitful discussions.

REFERENCES

- [1] R. Alur, T. Henzinger, and O. Kupferman. Alternating-Time Temporal Logic. *Journal of the ACM*, 49(5):672–713, 2002.
- [2] D. Berwanger, A. B. Mathew, and M. van den Bogaard. Hierarchical Information Patterns and Distributed Strategy Synthesis. In *ATVA’15*, LNCS 9364, pages 378–393. Springer, 2015.
- [3] N. Bulling and W. Jamroga. Comparing variants of strategic ability: how uncertainty and memory influence general properties of games. *Journal of Autonomous agents and multi-agent systems*, 28(3):474–518, 2014.
- [4] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. MIT Press, 1999.
- [5] C. Dima, C. Enea, and D. Guelev. Model-checking an alternating-time temporal logic with knowledge, imperfect information, perfect recall and communicating coalitions. In *GandALF 2010*, volume 25 of *EPTCS*, pages 103–117, 2010.
- [6] C. Dima and F. L. Tiplea. Model-checking ATL under imperfect information and perfect recall semantics is undecidable. *CoRR*, abs/1102.4225, 2011.
- [7] D. Easley and J. Kleinberg. *Networks, Crowds, and Markets: Reasoning About a Highly Connected World*. Cambridge University Press, 2010.
- [8] R. Fagin, J. Halpern, Y. Moses, and M. Vardi. *Reasoning about Knowledge*. MIT Press, 1995.
- [9] B. Finkbeiner and S. Schewe. Uniform Distributed Synthesis. In *LICS’05*, pages 321–330. IEEE Computer Society, 2005.

- [10] V. Goranko and W. Jamroga. Comparing semantics of logics for multi-agent systems. *Synthese*, 139(2):241–280, 2004.
- [11] D. P. Guelev, C. Dima, and C. Enea. An alternating-time temporal logic with knowledge, perfect recall and past: axiomatisation and model-checking. *Journal of Applied Non-Classical Logics*, 21(1):93–131, 2011.
- [12] J. Gutierrez, P. Harrenstein, and M. Wooldridge. Reasoning about equilibria in game-like concurrent systems. In *KR’14*. AAAI, 2014.
- [13] J. Gutierrez, P. Harrenstein, and M. Wooldridge. Expressiveness and complexity results for strategic reasoning. In *CONCUR’15*, volume 42 of *LIPICS*, 2015.
- [14] W. Hoek and M. Wooldridge. Cooperation, knowledge, and time: Alternating-time temporal epistemic logic and its applications. *Studia Logica*, 75(1):125–157, 2003.
- [15] W. Jamroga and W. van der Hoek. Agents that know how to play. *Fundamenta Informaticae*, 62:1–35, 2004.
- [16] M. Jurdzinski. Small Progress Measures for Solving Parity Games. In *STACS’00*, LNCS 1770, pages 290–301. Springer, 2000.
- [17] P. Kazmierczak, T. Ågotnes, and W. Jamroga. Multi-agency is coordination and (limited) communication. In *PRIMA’14*, LNCS 8861, pages 91–106. Springer, 2014.
- [18] O. Kupferman and M. Vardi. Synthesizing Distributed Systems. In *LICS’01*, pages 389–398. IEEE Computer Society, 2001.
- [19] O. Kupferman, M. Vardi, and P. Wolper. An Automata Theoretic Approach to Branching-Time Model Checking. *Journal of the ACM*, 47(2):312–360, 2000.
- [20] O. Kupferman, M. Vardi, and P. Wolper. Module Checking. *Information and Computation*, 164(2):322–344, 2001.
- [21] F. Laroussinie, N. Markey, and A. Sangnier. ATLsc with partial observation. In *GandALF 2015*, volume 193 of *EPTCS*, pages 43–57, 2015.
- [22] A. Lomuscio, H. Qu, and F. Raimondi. MCMAS: A model checker for the verification of multi-agent systems. *Software Tools for Technology Transfer*, 19(1):9–30, 2017.
- [23] A. R. Lomuscio, R. van der Meyden, and M. Ryan. Knowledge in multiagent systems: Initial configurations and broadcast. *ACM Trans. Comput. Logic*, 1(2):247–284, Oct. 2000.
- [24] A. Lopes, F. Laroussinie, and N. Markey. ATL with Strategy Contexts: Expressiveness and Model Checking. In *FSTTCS’10*, LIPICS 8, pages 120–132. Leibniz-Zentrum fuer Informatik, 2010.
- [25] F. Mogavero, A. Murano, G. Perelli, and M. Vardi. Reasoning About Strategies: On the Model-Checking Problem. *TOCL*, 15(4):34:1–42, 2014.
- [26] H. P. Nii. Blackboard systems, part one: The blackboard model of problem solving and the evolution of blackboard architectures. *AI Magazine*, 7(2):38–53, 1986.
- [27] A. Pnueli and R. Rosner. Distributed Reactive Systems Are Hard to Synthesize. In *FOCS’90*, pages 746–757. IEEE Computer Society, 1990.
- [28] R. Rosner. *Modular Synthesis of Reactive Systems*. PhD thesis, Weizmann Institute of Science, Rehovot, Israel, 1992.
- [29] S. Safra. *Complexity of Automata on Infinite Objects*. PhD thesis, Weizmann Institute of Science, Rehovot, Israel, 1989.
- [30] S. Schewe and B. Finkbeiner. Distributed synthesis for alternating-time logics. In K. S. Namjoshi, T. Yoneda, T. Higashino, and Y. Okamura, editors, *ATVA’07*, pages 268–283. Springer, 2007.
- [31] P. Schobbens. Alternating-Time Logic with Imperfect Recall. *ENTCS*, 85(2):82–93, 2004.
- [32] R. van der Meyden and H. Shilov. Model checking knowledge and time in systems with perfect recall. In *FST&TCS’99*, LNCS 1738, pages 432–445. Springer, 1999.
- [33] R. van der Meyden and T. Wilke. Synthesis of distributed systems from knowledge-based specifications. In *CONCUR’05*, LNCS 3653, pages 562–576. Springer, 2005.
- [34] H. van Ditmarsch, W. van der Hoek, and B. Kooi. *Dynamic Epistemic Logic*, volume 337 of *Synthese Library*. Springer, 2007.
- [35] M. Y. Vardi and P. Wolper. Reasoning about infinite computations. *Information and Computation*, 115(1):1–37, 1994.
- [36] S. Vester. Alternating-time temporal logic with finite-memory strategies. In *GandALF’13*, volume 119 of *EPTCS*, pages 194–207, 2013.

Strategy Logic with Imperfect Information

Raphaël Berthon*, Bastien Maubert†, Aniello Murano†, Sasha Rubin† and Moshe Y. Vardi‡

*École Normale Supérieure de Rennes, Rennes, France

†Università degli Studi di Napoli Federico II, Naples, Italy

‡Rice University, Houston, Texas, USA

Abstract—We introduce an extension of Strategy logic for the imperfect-information setting, called SL_{ii} , and study its model-checking problem. As this logic naturally captures multi-player games with imperfect information, the problem turns out to be undecidable. We introduce a syntactical class of “hierarchical instances” for which, intuitively, as one goes down the syntactic tree of the formula, strategy quantifications are concerned with finer observations of the model. We prove that model-checking SL_{ii} restricted to hierarchical instances is decidable. This result, because it allows for complex patterns of existential and universal quantification on strategies, greatly generalises previous ones, such as decidability of multi-player games with imperfect information and hierarchical observations, and decidability of distributed synthesis for hierarchical systems.

To establish the decidability result, we introduce and study $\text{QCTL}_{\text{ii}}^*$, an extension of QCTL (itself an extension of CTL with second-order quantification over atomic propositions) by parameterising its quantifiers with observations. The simple syntax of $\text{QCTL}_{\text{ii}}^*$ allows us to provide a conceptually neat reduction of SL_{ii} to $\text{QCTL}_{\text{ii}}^*$ that separates concerns, allowing one to forget about strategies and players and focus solely on second-order quantification. While the model-checking problem of $\text{QCTL}_{\text{ii}}^*$ is, in general, undecidable, we identify a syntactic fragment of hierarchical formulas and prove, using an automata-theoretic approach, that it is decidable. The decidability result for SL_{ii} follows since the reduction maps hierarchical instances of SL_{ii} to hierarchical formulas of $\text{QCTL}_{\text{ii}}^*$.

I. INTRODUCTION

Logics for strategic reasoning are a powerful tool for expressing correctness properties of multi-player graph-games, which in turn are natural models for reactive systems and discrete event systems. In particular, ATL^* and its related logics were introduced to capture the realisability/synthesis problem for open systems with multiple components. These logics were designed as extensions of branching-time logics such as CTL^* that allow one to write alternating properties directly in the syntax, i.e., statements of the form “there exist strategies, one for each player in A , such for all strategies of the remaining players, the resulting play satisfies φ ”. Strategy logic (SL) [1] generalises these by treating strategies as first-order objects x that can be quantified $\langle\langle x\rangle\rangle$ (read “there exists a strategy x ”) and bound to players (a, x) (read “player a uses strategy x ”). This syntax has flexibility very similar to first-order logic, and thus allows one to directly express many solution concepts from game-theory, e.g., the SL formula $\langle\langle x_1\rangle\rangle\langle\langle x_2\rangle\rangle(a_1, x_1)(a_2, x_2) \wedge_{i=1,2} [\langle\langle y_i\rangle\rangle(a_i, y_i)\text{goal}_i] \rightarrow \text{goal}_1$ expresses the existence of a Nash equilibrium in a two-player game (with individual Boolean objectives).

An essential property of realistic multi-player games is that players only have a limited view of the state of the system. This is captured by introducing partial-observability into the models, i.e., equivalence-relations o (called *observations*) over the state space that specify indistinguishable states. In the formal-methods literature it is typical to associate observations to players. In this paper, instead, we associate observations to strategies. Concretely, we introduce an extension SL_{ii} of SL that annotates the strategy quantifier $\langle\langle x\rangle\rangle$ by an observation o , written $\langle\langle x\rangle\rangle^o$. Thus, both the model and the formulas mention observations o . This novelty allows one to express, in the logic, that a player’s observation changes over time.

Our logic SL_{ii} is extremely powerful: it is an extension of SL (which is in the perfect-information setting), and of the imperfect-information strategic logics $\text{ATL}_{\text{i},\text{R}}^*$ [2] and $\text{ATL}_{\text{sc},\text{i}}^*$ [3]. A canonical specification in multi-player game of partial observation is that the players, say a_1, \dots, a_n , can form a coalition and beat the environment player, say b . This can be expressed in SL_{ii} as $\Phi_{\text{SYNTH}} := \langle\langle x_1\rangle\rangle^{o_1} \dots \langle\langle x_n\rangle\rangle^{o_n} \llbracket y \rrbracket^o (a_1, x_1) \dots (a_n, x_n)(b, y)\varphi$, where φ is quantifier- and binding-free. Also, SL_{ii} can express more complicated specifications by alternating quantifiers, binding the same strategy to different agents and rebinding (these are inherited from SL), as well as changing observations.

The complexity of SL_{ii} is also visible from an algorithmic point of view. Its satisfiability problem is undecidable (this is already true of SL), and its model-checking problem is undecidable (this is already true of $\text{ATL}_{\text{i},\text{R}}^*$, in fact, even for the single formula $\langle\{a, b\}\rangle \text{F}p$ in systems with three players [4]). In fact, similar undecidability occurs in foundational work in multi-player games of partial observation, and in distributed synthesis [5], [6]. Since then, the formal-methods community has spent much effort finding restrictions and variations that ensure decidability [7]–[13]. The common thread in these approaches is that the players’ observations (or what they can deduce from their observations) are hierarchical.

Motivated by the problem of finding decidable extensions of strategy logic in the imperfect-information setting, we introduce a syntactical class of “hierarchical instances” of SL_{ii} , i.e., formula/model pairs, and prove that the model-checking problem on this class of instances is decidable. Intuitively, an instance of SL_{ii} is hierarchical if, as one goes down the syntactic tree of the formula, the observations annotating strategy quantifications can only become finer. Although the class of hierarchical instances refers not only to the syntax of the logic but also to the model, the class is syntactical in the

sense that it depends only on the structure of the formula and the observations in the model. Moreover, it is easy to check (i.e., in linear time) if an instance is hierarchical or not.

The class of hierarchical instances generalises some existing approaches and supplies new classes of systems and properties that can be model-checked. For instance, suppose that there is a total order \preceq among the players such that $a \preceq b$ implies player b 's observation is finer than player a 's observation — such games are said to yield “hierarchical observation” in [13]. In such games it is known that synthesis against CTL^* specifications is decidable (in fact, this holds for ω -regular specifications [8], [13]). This corresponds to hierarchical instances of SL_{ii} in which the observations form a total order in the model and the formula is of the form Φ_{SYNTH} (mentioned above). On the other hand, in hierarchical instances of SL_{ii} , the ordering on observations can be a pre partial-order (i.e., not just total), and one can arbitrarily alternate quantifiers in the formulas. For instance, hierarchical instances allow one to decide if a game that yields hierarchical information has a Nash equilibrium. For example, assuming observations p_1, p_2, o_1, o_2 with p_i finer than o_2 (for $i = 1, 2$), and o_2 finer than o_1 , the formula $\langle\langle x_1 \rangle\rangle^{o_1} \langle\langle x_2 \rangle\rangle^{o_2} (a_1, x_1)(a_2, x_2) \wedge_{i=1,2} [\langle\langle y_i \rangle\rangle^{p_i} (a_i, y_i) goal_i] \rightarrow goal_i$ expresses that there exists a strategy profile (x_1, x_2) of uniform strategies, such that neither player has an incentive to deviate using a strategy that observes at least as much as the observations that both players started with. Observe that this formula is in fact *equivalent* to the existence of a Nash equilibrium, i.e., to the same formula in which $p_i = o_i$. This shows we can decide the existence of Nash equilibria in games that yield hierarchical observation.

In order to reason about SL_{ii} we introduce an intermediate logic QCTL_{ii}^* , an extension to the imperfect-information setting of QCTL^* [14], itself an extension of CTL^* by second-order quantifiers over atoms. This is a low-level logic that does not mention strategies and into which one can effectively compile instances of SL_{ii} . States of the models of the logic QCTL_{ii}^* have internal structure, much like the multi-player game structures from [15] and distributed systems [16]. Model-checking QCTL_{ii}^* is also undecidable (indeed, we show how to reduce from the MSO-theory of the binary tree extended with the equal-length predicate, known to be undecidable [17]). We introduce the syntactical class $\text{QCTL}_{i,\subseteq}^*$ of hierarchical formulas as those in which innermost quantifiers observe more than outermost quantifiers, and prove that model-checking is decidable using an extension of the automata-theoretic approach for branching-time logics (our decision to base models of QCTL_{ii}^* on local states greatly eases the use of automata). Moreover, the compilation of SL_{ii} into QCTL_{ii}^* preserves being hierarchical, thus establishing our main contribution, i.e., that model checking the hierarchical instances of SL_{ii} is decidable.

Related work. Formal methods for reasoning about reactive systems with multiple components have been studied mainly in two theoretical frameworks: a) multi-player graph-games of partial-observation [6], [8], [13] and b) synthesis in distributed architectures [5], [7], [9], [11], [18] (the relationship

between these two frameworks is discussed in [13]). All of these works consider the problem of synthesis, which (for objectives in temporal logics) can be expressed in SL_{ii} using the formula Φ_{SYNTH} mentioned above. Limited alternation was studied in [12] that, in the language of SL_{ii} , considers the model-checking problem of formulas of the form $\langle\langle x_1 \rangle\rangle^{o_1} \llbracket x_2 \rrbracket^{o_2} \langle\langle x_3 \rangle\rangle^{o_3} (a_1, x_1)(a_2, x_2)(a_3, x_3)\varphi$, where φ is an ω -regular objective. They prove that this is decidable in case player player 3 has perfect observation and player 2 observes at least as much as player 1.

In contrast to all these works, formulas of SL_{ii} can express much more complex specifications by alternating quantifiers, sharing strategies, rebinding, and changing observations.

Recently, [13] generalised the classic result of [8]: it weakens the assumption of hierarchical observation to hierarchical information (which are both static notions), and then, further to dynamic hierarchical information which allows for the hierarchy amongst players' information to change along a play. However, they only consider the synthesis problem.

We are aware of two papers that (like we do) give simultaneous structural constraints on both the formula and the model that result in decidability: in the context of synthesis in distributed architecture with process delays, [18] considers CTL^* specifications that constrain external variables by the input variables that may effect them in the architecture; and in the context of asynchronous perfect-recall, [10] considers a syntactical restriction on instances for Quantified μ -Calculus with partial observation (in contrast, we consider the case of synchronous perfect recall).

The work closest to ours is [19] which introduces a decidable logic CL in which one can encode many distributed synthesis problems. However, CL is close in spirit to our $\text{QCTL}_{i,\subseteq}^*$, and is more appropriate as a tool than as a high-level specification logic like SL_{ii} . Furthermore, by means of a natural translation we derive that CL is strictly included in the hierarchical instances of SL_{ii} (Section II-E). In particular, we find that hierarchical instances of SL_{ii} can express non-observable goals, while CL does not. Non-observable goals arise naturally in problems in distributed synthesis [5].

Finally, our logic SL_{ii} is the first generalisation of (the syntax and semantics of) SL to include strategies with partial observation, and, unlike CL , generalises previous logics with partial-observation strategies, i.e., $\text{ATL}_{i,R}^*$ [2] and $\text{ATL}_{sc,i}^*$ [3]. A comparison of SL_{ii} to SL , CL , and $\text{ATL}_{sc,i}^*$ is given in Section II-E.

Plan. The definition of SL_{ii} and of hierarchical instances, and the discussion about Nash equilibria, are in Section II. The definition of QCTL_{ii}^* , and its hierarchical fragment $\text{QCTL}_{i,\subseteq}^*$, are in Section III. The proof that model checking $\text{QCTL}_{i,\subseteq}^*$ is decidable, including the required automata preliminaries, are in Section IV. The translation of SL_{ii} into QCTL_{ii}^* , and the fact that this preserves hierarchy, are in Section V. Missing details are in the Appendix.

II. SL WITH IMPERFECT INFORMATION

In this section we introduce SL_{ii} , an extension of SL [1] to the imperfect-information setting with synchronous perfect-recall. Our logic presents two original features: first, observations are not bound to players (as is done in extensions of ATL by imperfect information [20] or logics for reasoning about knowledge [21]), and second, we have syntactic observations in the language, which need to be interpreted.

We follow the presentation of SL in [1], except that we make some changes that simplify their presentation but do not change their semantics, and some that allow us to capture imperfect information. We introduce the syntax and semantics of SL_{ii} , carefully detailing these changes. We first fix some notation used throughout the paper.

A. Notation

Let Σ be an alphabet. A *finite* (resp. *infinite*) *word* over Σ is an element of Σ^* (resp. Σ^ω). Words are written $w = w_0w_1w_2\dots$, i.e., indexing begins with 0. The *length* of a finite word $w = w_0w_1\dots w_n$ is $|w| := n + 1$, and $\text{last}(w) := w_n$ is its last letter. Given a finite (resp. infinite) word w and $0 \leq i \leq |w|$ (resp. $i \in \mathbb{N}$), we let w_i be the letter at position i in w , $w_{\leq i}$ is the prefix of w that ends at position i and $w_{\geq i}$ is the suffix of w that starts at position i . We write $w \preccurlyeq w'$ if w is a prefix of w' , and w^\preccurlyeq is the set of finite prefixes of word w . Finally, the domain of a mapping f is written $\text{dom}(f)$, and for $n \in \mathbb{N}$ we let $[n] := \{i \in \mathbb{N} : 1 \leq i \leq n\}$. The literature sometimes refers to “imperfect information” and sometimes to “partial observation”; we will use the terms interchangeably.

B. Syntax

The syntax of SL_{ii} is similar to that of strategy logic SL in [1]: the only difference is that we annotate strategy quantifiers $\langle\!\langle x \rangle\!\rangle$ by *observation symbols* o . For the rest of the paper, for convenience we fix a number of parameters for our logics and models: AP is a finite set of *atomic propositions*, Ag is a finite set of *players*, Var is a finite set of *variables* and Obs is a finite set of *observation symbols*. When we consider model-checking problems, these data are implicitly part of the input.

Definition 1 (SL_{ii} Syntax). *The syntax of SL_{ii} is defined by the following grammar:*

$$\varphi := p \mid \neg\varphi \mid \varphi \vee \varphi \mid \mathbf{X}\varphi \mid \varphi \mathbf{U}\varphi \mid \langle\!\langle x \rangle\!\rangle^o \varphi \mid (a, x)\varphi$$

where $p \in \text{AP}$, $x \in \text{Var}$, $o \in \text{Obs}$ and $a \in \text{Ag}$.

We use abbreviations $\top := p \vee \neg p$, $\perp := \neg\top$, $\varphi \rightarrow \varphi' := \neg\varphi \vee \varphi'$, $\varphi \leftrightarrow \varphi' := \varphi \rightarrow \varphi' \wedge \varphi' \rightarrow \varphi$ for boolean connectives, $\mathbf{F}\varphi := \top \mathbf{U}\varphi$, $\mathbf{G}\varphi := \neg\mathbf{F}\neg\varphi$ for temporal operators, and finally $\langle\!\langle x \rangle\!\rangle^o \varphi := \neg\langle\!\langle x \rangle\!\rangle^o \neg\varphi$.

The notion of free variables and sentences are defined as for SL . We recall these for completeness (formal definitions are in the appendix). A variable x appears *free* in a formula φ if it appears out of the scope of a strategy quantifier, and a player a appears free in φ if a temporal operator (either \mathbf{X} or \mathbf{U}) appears in φ out of the scope of any binding for player a . We let $\text{free}(\varphi)$ be the set of variables and players that appear free in φ . If $\text{free}(\varphi)$ is empty, φ is a *sentence*.

C. Semantics

The models of SL_{ii} are like those of SL , i.e., concurrent game structures, but extended by a finite set of observations Obs and, for each $o \in \text{Obs}$, by an equivalence-relation $\mathcal{O}(o)$ over positions that represents what a player using a strategy with that observation can see. That is, $\mathcal{O}(o)$ -equivalent positions are indistinguishable to a player using a strategy associated with observation o .

Definition 2 (CGS_{ii}). *A concurrent game structure with imperfect information (or CGS_{ii} for short) $\mathcal{G} = (\text{Ac}, V, E, \ell, v_\iota, \mathcal{O})$ where*

- Ac is a finite non-empty set of actions,
- V is a finite non-empty set of positions,
- $E : V \times \text{Ac}^{\text{Ag}} \rightarrow V$ is a transition function,
- $\ell : V \rightarrow 2^{\text{AP}}$ is a labelling function,
- $v_\iota \in V$ is an initial position,
- $\mathcal{O} : \text{Obs} \rightarrow V \times V$ is an observation interpretation mapping observations to equivalence relations on positions.

We may write \sim_o for $\mathcal{O}(o)$, and $v \in \mathcal{G}$ for $v \in V$.

The notions of joint actions, plays, strategies and assignments are similar to those for SL . We will recall these for completeness. Since the main difference between the semantics of SL and SL_{ii} is in the strategy-quantification case where we require strategies to be consistent with observations, we define synchronous perfect recall and o -strategies before defining the semantics of SL_{ii} . Finally, we mention that we make some (inconsequential) simplifications to the semantics of SL as proposed in [1]: i) we dispense with partial assignments and only consider complete assignments, ii) our semantics are w.r.t. a finite play instead of a position (in the Appendix we prove that these simplifications do not change the expressive power of SL).

Joint actions. In a position $v \in V$, each player a chooses an action $c_a \in \text{Ac}$, and the game proceeds to position $E(v, c)$, where $c \in \text{Ac}^{\text{Ag}}$ stands for the *joint action* $(c_a)_{a \in \text{Ag}}$. Given a joint action $c = (c_a)_{a \in \text{Ag}}$ and $a \in \text{Ag}$, we let c_a denote c_a . For each position $v \in V$, $\ell(v)$ is the set of atomic propositions that hold in v .

Plays and strategies. A *finite* (resp. *infinite*) *play* is a finite (resp. infinite) word $\rho = v_0 \dots v_n$ (resp. $\pi = v_0 v_1 \dots$) such that $v_0 = v_\iota$ and for all i with $0 \leq i < |\rho| - 1$ (resp. $i \geq 0$), there exists a joint action c such that $E(v_i, c) = v_{i+1}$. We let Plays be the set of finite plays. A *strategy* is a function $\sigma : \text{Plays} \rightarrow \text{Ac}$, and the set of all strategies is denoted Str .

Assignments. An *assignment* is a function $\chi : \text{Ag} \cup \text{Var} \rightarrow \text{Str}$, assigning a strategy to each player and variable. For an assignment χ , player a and strategy σ , $\chi[a \mapsto \sigma]$ is the assignment that maps a to σ and is equal to χ on the rest of its domain, and $\chi[x \mapsto \sigma]$ is defined similarly, where x is a variable.

Outcomes. For an assignment χ and a finite play ρ , we let $\text{out}(\chi, \rho)$ be the only infinite play that starts with ρ and is then extended by letting players follow the strategies assigned by χ .

Formally, $\text{out}(\chi, \rho) := \rho \cdot v_1 v_2 \dots$ where, for all $i \geq 0$, $v_{i+1} = E(v_i, c)$, where $v_0 = \text{last}(\rho)$ and $c = (\chi(a)(\rho \cdot v_1 \dots v_i))_{a \in \text{Ag}}$.

Synchronous perfect recall. In this work we consider players with *synchronous perfect recall*, meaning that each player remembers the whole history of a play, a classic assumption in games with imperfect information and logics of knowledge and time. Each observation relation is thus extended to finite plays as follows: $\rho \sim_o \rho'$ if $|\rho| = |\rho'|$ and $\rho_i \sim_o \rho'_i$ for every $i \in \{0, \dots, |\rho| - 1\}$. For $o \in \text{Obs}$, an o -strategy is a strategy $\sigma : V^+ \rightarrow \text{Ac}$ such that $\sigma(\rho) = \sigma(\rho')$ whenever $\rho \sim_o \rho'$. The latter constraint captures the essence of imperfect information, which is that players can base their strategic choices only on the information available to them. For $o \in \text{Obs}$ we let Str_o be the set of all o -strategies.

Definition 3 (SL_{ii} Semantics). *The semantics $\mathcal{G}, \chi, \rho \models \varphi$ is defined inductively, where φ is an SL_{ii} -formula, $\mathcal{G} = (\text{Ac}, V, E, \ell, v_\iota, \mathcal{O})$ is a CGS_{ii}, ρ is a finite play, and χ is an assignment:*

$$\begin{aligned} \mathcal{G}, \chi, \rho \models p &\quad \text{if } p \in \ell(\text{last}(\rho)) \\ \mathcal{G}, \chi, \rho \models \neg\varphi &\quad \text{if } \mathcal{G}, \chi, \rho \not\models \varphi \\ \mathcal{G}, \chi, \rho \models \varphi \vee \varphi' &\quad \text{if } \mathcal{G}, \chi, \rho \models \varphi \text{ or } \mathcal{G}, \chi, \rho \models \varphi' \\ \mathcal{G}, \chi, \rho \models \langle\langle x \rangle\rangle^{o_2} \varphi &\quad \text{if } \exists \sigma \in \text{Str}_o \text{ s.t. } \mathcal{G}, \chi[x \mapsto \sigma], \rho \models \varphi \\ \mathcal{G}, \chi, \rho \models (a, x)\varphi &\quad \text{if } \mathcal{G}, \chi[a \mapsto \chi(x)], \rho \models \varphi \end{aligned}$$

and, writing $\pi = \text{out}(\chi, \rho)$:

$$\begin{aligned} \mathcal{G}, \chi, \rho \models \mathbf{X}\varphi &\quad \text{if } \mathcal{G}, \chi, \pi^{\leq |\rho|} \models \varphi \\ \mathcal{G}, \chi, \rho \models \varphi \mathbf{U} \varphi' &\quad \text{if } \exists i \geq 0 \text{ s.t. } \mathcal{G}, \chi, \pi^{\leq |\rho| + i - 1} \models \varphi' \\ &\quad \text{and } \forall j \text{ s.t. } 0 \leq j < i, \\ &\quad \mathcal{G}, \chi, \pi^{\leq |\rho| + j - 1} \models \varphi \end{aligned}$$

To explain the temporal operators, we remind the reader that positions begin at 0 and thus π_n is the $(n+1)$ -st position of π . The satisfaction of a sentence is independent of the assignment (the easy proof is in the Appendix). For an SL_{ii} sentence φ we thus let $\mathcal{G}, \rho \models \varphi$ if $\mathcal{G}, \chi, \rho \models \varphi$ for some assignment χ , and we write $\mathcal{G} \models \varphi$ if $\mathcal{G}, v_\iota \models \varphi$.

D. Model checking and hierarchical instances

Model Checking. We now introduce the main decision problem of this paper, i.e., the model-checking problem for SL_{ii} . An SL_{ii} -instance is a formula/model pair (Φ, \mathcal{G}) where $\Phi \in \text{SL}_{ii}$ and \mathcal{G} is a CGS_{ii}. The *model-checking problem* for SL_{ii} is the decision problem that, given an SL_{ii} -instance (Φ, \mathcal{G}) , returns ‘yes’ if $\mathcal{G} \models \Phi$, and ‘no’ otherwise.

It is well known that deciding the existence of winning strategies in multi-player games with imperfect information is undecidable for reachability objectives [15]. Since this problem is easily reduced to the model-checking problem for SL_{ii} , we get the following result.

Theorem 1. *The model-checking problem for SL_{ii} is undecidable.*

Hierarchical instances. We now isolate a sub-problem obtained by restricting attention to *hierarchical instances*. Intuitively, an SL_{ii} -instance (Φ, \mathcal{G}) is hierarchical if, as one goes down a path in the syntactic tree of Φ , the observations tied

to quantifications become finer. We now make these notions precise.

Definition 4 (Hierarchical instances). *An SL_{ii} -instance (Φ, \mathcal{G}) is hierarchical if for all subformulas φ_1, φ_2 of Φ of the form $\varphi_2 = \langle\langle x \rangle\rangle^{o_2} \varphi'_2$ and $\varphi_1 = \langle\langle y \rangle\rangle^{o_1} \varphi'_1$ where φ_1 is a subformula of φ'_2 , it holds that $\mathcal{O}(\varphi_1) \subseteq \mathcal{O}(\varphi_2)$.*

If $\mathcal{O}(\varphi_1) \subseteq \mathcal{O}(\varphi_2)$ we say that φ_1 is *finer* than φ_2 in \mathcal{G} , and that φ_2 is *coarser* than φ_1 in \mathcal{G} . Intuitively, this means that a player with observation φ_1 observes game \mathcal{G} no worse than, i.e., is not less informed, a player with observation φ_2 .

Example 1 (Security levels). *We illustrate hierarchical instances in a “security levels” scenario, e.g., higher levels have access to more data (i.e., can observe more). Assume that the CGS_{ii} \mathcal{G} has $\mathcal{O}(o_3) \subseteq \mathcal{O}(o_2) \subseteq \mathcal{O}(o_1)$ (i.e., level 3 has the highest security clearance, while level 1 has the lowest). Let $\varphi = (a_1, x_1)(a_2, x_2)(a_3, x_3)\mathbf{G}p$. The SL_{ii} formula $\Phi := \langle\langle x_1 \rangle\rangle^{o_1} \llbracket x_2 \rrbracket^{o_2} \langle\langle x_3 \rangle\rangle^{o_3} \varphi$ forms a hierarchical instance when paired with \mathcal{G} . It expresses that the player a_1 (with lowest clearance) can collude with player a_3 (having the highest clearance) to ensure a safety property p , even in the presence of an adversary a_2 (with intermediate clearance), as long as the strategy used by a_3 can also depend on the strategy used by a_2 .*

On the other hand, the similar formula $\langle\langle x_1 \rangle\rangle^{o_1} \langle\langle x_3 \rangle\rangle^{o_3} \llbracket x_2 \rrbracket^{o_2} \varphi$, which implies that the strategy used by a_3 cannot depend on the adversarial strategy used by a_2 , does not form a hierarchical instance when paired with \mathcal{G} .

Here is the main contribution of this paper:

Theorem 2. *The model-checking problem for SL_{ii} restricted to the class of hierarchical instances is decidable.*

This is proved in Section V by reducing it to the model-checking problem of the hierarchical fragment of a logic called QCTL_{ii}^{*}, which we introduce, and prove decidable, in Section III. We now give an important corollary of the main theorem.

A Nash equilibrium in a game is a tuple of strategies such that no player has the incentive to deviate. Assuming that goals are written in SL_{ii} , say goal_i for $i \in \text{Ag}$, and $\text{Ag} = \{a_i : i \in [n]\}$, the following formula of SL_{ii} expresses the existence of a Nash equilibrium:

$$\Phi_{\text{NE}} := \langle\langle x_1 \rangle\rangle^{o_1} \dots \langle\langle x_n \rangle\rangle^{o_n} (a_1, x_1) \dots (a_n, x_n) \Psi_{\text{NE}}$$

where $\Psi_{\text{NE}} := \bigwedge_{i \in [n]} [(\langle\langle y_i \rangle\rangle^{o_i} (a_i, y_i) \text{goal}_i) \rightarrow \text{goal}_i]$.

A CGS_{ii} \mathcal{G} is said to *yield hierarchical observation* [13] if the “finer-than” relation is a total ordering, i.e., if for all $o, o' \in \text{Obs}$, either $\mathcal{O}(o) \subseteq \mathcal{O}(o')$ or $\mathcal{O}(o') \subseteq \mathcal{O}(o)$.

Note that the instance $(\Phi_{\text{NE}}, \mathcal{G})$ is *not* hierarchical (unless $\mathcal{O}(\varphi_i) = \mathcal{O}(\varphi_j)$ for all $i, j \in \text{Ag}$). Nonetheless, we can decide if a game that yields hierarchical observation has a Nash equilibrium:

Corollary 1. *The following problem is decidable: given a CGS_{ii} that yields hierarchical observation, whether $\mathcal{G} \models \Phi_{\text{NE}}$.*

Proof. The main idea is to use the fact that in a one-player game of partial-observation (such a game occurs when all but one player have fixed their strategies, as in the definition of Nash equilibrium), the player has a strategy enforcing some goal iff the player has a uniform-strategy enforcing that goal. Here are the details. Let $\mathcal{G} = (\text{Ac}, V, E, \ell, v_\ell, \mathcal{O})$ be a CGS_{ii} that yields hierarchical observation. Suppose the observation set is Obs. To decide if $\mathcal{G} \models \Phi_{\text{NE}}$ first build a new CGS_{ii} $\mathcal{G}' = (\text{Ac}, V, E, \ell, v_\ell, \mathcal{O}')$ over observations Obs' := Obs $\cup \{o_p\}$ such that $\mathcal{O}'(o) = \mathcal{O}(o)$ and $\mathcal{O}'(o_p) = \{(v, v) : v \in V\}$, and consider the sentence

$$\Phi' := \langle\langle x_1 \rangle\rangle^{o_1} \dots \langle\langle x_n \rangle\rangle^{o_n} (a_1, x_1) \dots (a_n, x_n) \Psi'$$

where $\Psi' := \bigwedge_{i \in [n]} [(\langle\langle y_i \rangle\rangle^{o_p} (a_i, y_i) \text{goal}_i) \rightarrow \text{goal}_i]$.

Then (Φ', \mathcal{G}') is a hierarchical instance, and by Theorem 2 we can decide $\mathcal{G}' \models \Phi'$. We claim that $\mathcal{G}' \models \Phi'$ iff $\mathcal{G} \models \Phi_{\text{NE}}$. To see this, it is enough to establish that:

$$\mathcal{G}', \chi, v_\ell \models \langle\langle y_i \rangle\rangle^{o_p} (a_i, y_i) \text{goal}_i \leftrightarrow \langle\langle y_i \rangle\rangle^{o_i} (a_i, y_i) \text{goal}_i,$$

for every $i \in [n]$ and every assignment χ such that $\chi(x_i) = \chi(a_i)$ is an o_i -uniform strategy.

To this end, fix i and χ . The right-to-left implication is immediate (since o_p is finer than o_i). For the converse, let σ be a p -uniform strategy (i.e., perfect-information) such that $\mathcal{G}', \chi[y_i \mapsto \sigma, a_i \mapsto \sigma], v_\ell \models \text{goal}_i$. Let $\pi := \text{out}(\chi[y_i \mapsto \sigma, a_i \mapsto \sigma], v_\ell)$. Construct an o_i -uniform strategy σ' that agrees with σ on prefixes of π . This can be done as follows: if $\rho \sim_{o_i} \pi_{\leq j}$ for some j then define $\sigma'(\rho) = \sigma(\pi_{\leq j})$ (note that this is well-defined since if there is some such j then it is unique), and otherwise define $\sigma'(\rho) = a$ for some fixed action $a \in \text{Ac}$. \square

E. Comparison with other logics

The main difference between SL and ATL-like strategic logics is that in the latter a strategy is always bound to some player, while in the former bindings and quantifications are separated. This separation adds expressive power, e.g., one can bind the same strategy to different players. Extending ATL with imperfect-information is done by giving each player an indistinguishability relation that its strategies must respect [2]. Our extension of SL by imperfect information, instead, assigns each strategy x an indistinguishability relation o when it is quantified $\langle\langle x \rangle\rangle^o$. Thus $\langle\langle x \rangle\rangle^o \varphi$ means ‘‘there exists a strategy with observation o such that φ holds’’. Associating observations in this way, i.e., to strategies rather than players has two consequences. First, it is a clean generalisation of SL in the perfect information setting [1]. Define the *perfect-information fragment of SL_{ii}* to be the logic SL_{ii} assuming that Obs = $\{o\}$ and $\mathcal{O}(o) = \{(v, v) : v \in \mathcal{G}\}$. The next proposition says that the perfect-information fragment of SL_{ii} is a notational variant of SL [1] (the proof is in the Appendix).

Proposition 1. *For every SL sentence φ there is an SL_{ii} sentence φ' with Obs = $\{o\}$, such that for every CGS \mathcal{G} there is a CGS_{ii} \mathcal{G}' with $\mathcal{O}(o) = \{(v, v) : v \in \mathcal{G}\}$ such that $\mathcal{G} \models \varphi$ iff $\mathcal{G}' \models \varphi'$.*

Second, SL_{ii} subsumes imperfect-information extensions of ATL^{*} that associate observations to players.

Proposition 2. *For every ATL_{i,R}^{*} formula¹ φ there is an SL_{ii} formula φ' such that for every CGS_{ii} \mathcal{G} there is a CGS_{ii} \mathcal{G}' such that $\mathcal{G} \models \varphi$ iff $\mathcal{G}' \models \varphi'$.*

We recall that an ATL_{i,R}^{*} formula $\langle A \rangle \psi$ reads as ‘‘there are strategies for players in A such that ψ holds whatever players in $\text{Ag} \setminus A$ do’’. Formula φ' is built from φ by replacing each subformula of the form $\langle A \rangle \psi$, where $A = \{a_1, \dots, a_k\} \subset \text{Ag}$ is a coalition of players and $\text{Ag} \setminus A = \{a_{k+1}, \dots, a_n\}$, with formula $\langle\langle x_1 \rangle\rangle^{o_1} \dots \langle\langle x_k \rangle\rangle^{o_k} \llbracket x_{k+1} \rrbracket^{o_p} \dots \llbracket x_n \rrbracket^{o_p} (a_1, x_1) \dots (a_n, x_n) \psi'$, where ψ' is the translation of ψ . Then \mathcal{G}' is obtained from \mathcal{G} by interpreting each o_i as the equivalence relation for player i in \mathcal{G} , and interpreting o_p as the identity relation.

Third, SL_{ii} also subsumes the imperfect-information extension of ATL^{*} with strategy context (see [3] for the definition of ATL_{sc}^{*} with partial observation, which we refer to as ATL_{sc,i}^{*}).

Proposition 3. *For every ATL_{sc,i}^{*} formula φ there is an SL_{ii} formula φ' such that for every CGS_{ii} \mathcal{G} there is a CGS_{ii} \mathcal{G}' such that $\mathcal{G} \models \varphi$ iff $\mathcal{G}' \models \varphi'$.*

The only difference between ATL_{sc,i}^{*} and ATL_{i,R}^{*} is the following: in ATL_{i,R}^{*}, when a subformula of the form $\langle A \rangle \psi$ is met, we quantify existentially on strategies for players in A , and then we consider all possible outcomes obtained by letting other players behave however they want. Therefore, if any player in $\text{Ag} \setminus A$ had previously been assigned a strategy, it is forgotten. In ATL_{sc,i}^{*} on the other hand, these strategies are stored in a *strategy context*, which is a *partial* assignment χ , defined for the subset of players currently bound to a strategy. A strategy context allows one to quantify universally only on strategies of players who are not in A and who are not already bound to a strategy. It is then easy to adapt the translation presented for Proposition 2 by parameterising it with a strategy context. \mathcal{G}' is defined as for Proposition 2.

Fourth, there is a natural and simple translation of instances of the model-checking problem of CL [19] into the hierarchical instances of SL_{ii}. Moreover, the image of this translation consists of instances of SL_{ii} with a very restricted form, i.e., atoms mentioned in the SL_{ii}-formula are observable (by all observations of the CGS_{ii}), i.e., $v \sim_o v'$ implies $p \in \ell(v) \leftrightarrow p \in \ell(v')$.

Proposition 4. *There is an effective translation that, given a CL-instance (\mathcal{S}, φ) produces a hierarchical SL_{ii}-instance (\mathcal{G}, Φ) such that*

- 1) $\mathcal{S} \models \varphi$ iff $\mathcal{G} \models \Phi$,
- 2) *For all atoms p in Φ , and all observations $o \in \text{Obs}$, we have that $v \sim_o v'$ implies $p \in \ell(v) \leftrightarrow p \in \ell(v')$.*

To do this, one first translates CL into (hierarchical) QCTL_{ii}^{*}, the latter is defined in the next section. This step is a simple

¹See [2] for the definition of ATL_{i,R}^{*}, where subscript i refers to ‘‘imperfect information’’ and subscript R to ‘‘perfect recall’’. Also, we consider the so-called *objective semantics* for ATL_{i,R}^{*}.

reflection of the semantics of CL in that of $\text{QCTL}_{\text{ii}}^*$. Then one translates $\text{QCTL}_{\text{ii}}^*$ into SL_{ii} by a simple adaptation of the translation of QCTL^* into ATL_{sc}^* [22]. Details are in the Appendix.

III. QCTL * WITH IMPERFECT INFORMATION

In this section we introduce an imperfect-information extension of QCTL^* [14], [23]–[27]. In order to introduce imperfect information, instead of considering equivalence relations between states as in concurrent game structures, we will enrich Kripke structures by giving internal structure to their states, i.e., we see states as n -tuples of local states. This way of modelling imperfect information is inspired from Reif’s multiplayer game structures [15] and distributed systems [16], and we find it very suitable to application of automata techniques, as discussed in Section III-C.

The syntax of $\text{QCTL}_{\text{ii}}^*$ is similar to that of QCTL^* , except that we annotate second-order quantifiers by subsets $\mathbf{o} \subseteq [n]$. The idea is that quantifiers annotated by \mathbf{o} can only “observe” the local states indexed by $i \in \mathbf{o}$. We define the tree-semantics of $\text{QCTL}_{\text{ii}}^*$: this means that we interpret formulas on trees that are the unfoldings of Kripke structures (this will capture the fact that players in SL_{ii} have synchronous perfect recall).

We then define the syntactic class of *hierarchical formulas* and prove, using an automata-theoretic approach, that model checking this class of formulas is decidable.

A. QCTL $_{\text{ii}}^*$ Syntax

The syntax of $\text{QCTL}_{\text{ii}}^*$ is very similar to that of QCTL^* : the only difference is that we annotate quantifiers by a set of indices that defines the “observation” of that quantifier.

Definition 5 ($\text{QCTL}_{\text{ii}}^*$ Syntax). Fix $n \in \mathbb{N}$. The syntax of $\text{QCTL}_{\text{ii}}^*$ is defined by the following grammar:

$$\begin{aligned}\varphi &:= p \mid \neg\varphi \mid \varphi \vee \varphi \mid \mathbf{E}\psi \mid \exists^{\mathbf{o}} p. \varphi \\ \psi &:= \varphi \mid \neg\psi \mid \psi \vee \psi \mid \mathbf{X}\psi \mid \psi \mathbf{U}\psi\end{aligned}$$

where $p \in \text{AP}$ and $\mathbf{o} \subseteq [n]$.

Formulas of type φ are called *state formulas*, those of type ψ are called *path formulas*, and $\text{QCTL}_{\text{ii}}^*$ consists of all the state formulas defined by the grammar. We use standard abbreviation $\top := p \vee \neg p$, $\perp := \neg\top$, $\mathbf{F}\psi := \top \mathbf{U}\psi$, $\mathbf{G}\psi := \neg\mathbf{F}\neg\psi$ and $\mathbf{A}\psi := \neg\mathbf{E}\neg\psi$. We use $\exists p. \varphi$ as a shorthand for $\exists^{[n]} p. \varphi$. Finally we let $\forall p. \varphi := \neg\exists p. \neg\varphi$.

Given a $\text{QCTL}_{\text{ii}}^*$ formula φ , we define the set of *quantified propositions* $\text{AP}_{\exists}(\varphi) \subseteq \text{AP}$ as the set of atomic propositions p such that φ has a subformula of the form $\exists^{\mathbf{o}} p. \varphi$. We also define the set of *free propositions* $\text{AP}_f(\varphi) \subseteq \text{AP}$ as the set of atomic propositions that appear out of the scope of any quantifier of the form $\exists^{\mathbf{o}} p$. Observe that $\text{AP}_{\exists}(\varphi) \cap \text{AP}_f(\varphi)$ may not be empty in general, i.e., a proposition may appear both free and quantified in (different places of) a formula.

B. QCTL $_{\text{ii}}^*$ tree-semantics

We define the semantics on structures whose states are tuples of local states.

Local states. Let $\{L_i\}_{i \in [n]}$ denote $n \in \mathbb{N}$ disjoint finite sets of *local states*. For $I \subseteq [n]$, we let $L_I := \prod_{i \in I} L_i$ if $I \neq \emptyset$, and $L_\emptyset := \{\mathbf{0}\}$ where $\mathbf{0}$ is a special symbol.

Concrete observations. A set $\mathbf{o} \subseteq [n]$ is called a *concrete observation* (to distinguish it from observations o in the definitions of SL_{ii}).

Compound Kripke structures. These are like Kripke structures except that the states are elements of $L_{[n]}$. A *compound Kripke structure*, or CKS, over AP, is a tuple $\mathcal{S} = (S, R, s_t, \ell)$ where $S \subseteq L_{[n]}$ is a set of *states*, $R \subseteq S \times S$ is a left-total² *transition relation*, $s_t \in S$ is an *initial state*, and $\ell : S \rightarrow 2^{\text{AP}}$ is a *labelling function*.

A *path* in \mathcal{S} is an infinite sequence of states $\lambda = s_0s_1\dots$ such that for all $i \in \mathbb{N}$, $(s_i, s_{i+1}) \in R$. For $s \in S$, we let $\text{Paths}(s)$ be the set of all paths that start in s . A *finite path* is a finite non-empty prefix of a path. We may write $s \in \mathcal{S}$ for $s \in S$. Since we will interpret $\text{QCTL}_{\text{ii}}^*$ on unfoldings of CKS, we now define infinite trees.

Trees. In many works, trees are defined as prefix-closed sets of words with the empty word ϵ as root. Here trees represent unfoldings of Kripke structures, and we find it more convenient to see a node u as a sequence of states and the root as the initial state. Let X be a finite set (typically a set of states). An X -tree τ is a nonempty set of words $\tau \subseteq X^+$ such that:

- there exists $r \in X$, called the *root* of τ , such that each $u \in \tau$ starts with r ($r \preceq u$);
- if $u \cdot x \in \tau$ and $u \neq \epsilon$, then $u \in \tau$, and
- if $u \in \tau$ then there exists $x \in X$ such that $u \cdot x \in \tau$.

The elements of a tree τ are called *nodes*. If $u \cdot x \in \tau$, we say that $u \cdot x$ is a *child* of u . The *depth* of a node u is $|u|$. An X -tree τ is *complete* if $u \in \tau, x \in X$ implies $u \cdot x \in \tau$. A *path* in τ is an infinite sequence of nodes $\lambda = u_0u_1\dots$ such that for all $i \in \mathbb{N}$, u_{i+1} is a child of u_i , and $\text{Paths}(u)$ is the set of paths that start in node u . An *AP-labelled X-tree*, or (AP, X) -tree for short, is a pair $t = (\tau, \ell)$, where τ is an X -tree called the *domain* of t and $\ell : \tau \rightarrow 2^{\text{AP}}$ is a *labelling*. For a labelled tree $t = (\tau, \ell)$ and an atomic proposition $p \in \text{AP}$, we define the *p-projection* of t as the labelled tree $t \Downarrow_p := (\tau, \ell \Downarrow_p)$, where for each $u \in \tau$, $\ell \Downarrow_p(u) := \ell(u) \setminus \{p\}$. For a set of trees \mathcal{L} , we let $\mathcal{L} \Downarrow_p := \{t \Downarrow_p \mid t \in \mathcal{L}\}$. Finally, two labelled trees $t = (\tau, \ell)$ and $t' = (\tau', \ell')$ are *equivalent modulo p*, written $t \equiv_p t'$, if $t \Downarrow_p = t' \Downarrow_p$ (in particular, $\tau = \tau'$).

Narrowing. Let X and Y be two finite sets, and let $(x, y) \in X \times Y$. The X -narrowing of (x, y) is $(x, y) \downarrow_X := x$. This definition extends naturally to words over $X \times Y$ (pointwise), and thus to $X \times Y$ -trees.

For $J \subseteq I \subseteq [n]$ and $z = (l_i)_{i \in I} \in L_I$, we also define $z \downarrow_J := z \downarrow_{L_J}$, where z is seen as a pair $z = (z_1, z_2) \in L_J \times L_{I \setminus J}$. This is well defined because having taken sets L_i to be disjoint, the ordering of local states in z is indifferent. We also extend this definition to words and trees.

²i.e., for all $s \in S$, there exists s' such that $(s, s') \in R$.

Observe that when narrowing a tree, nodes with same narrowing are merged. In particular, for every L_I -tree τ , $\tau \downarrow \emptyset$ is the only L_\emptyset -tree, $\mathbf{0}^\omega$.

Quantification and uniformity. In $\text{QCTL}_{\text{ii}}^*$ the intuitive meaning of $\exists^{\mathbf{o}} p. \varphi$ in a tree t is that there is some equivalent tree t' modulo p such that t' is \mathbf{o} -uniform in p and satisfies φ . Intuitively, a tree is \mathbf{o} -uniform in p if it is uniformly labelled by p , i.e., if every two nodes that are indistinguishable when projected onto the local states indexed by $\mathbf{o} \subseteq [n]$ agree on their labelling of p .

Definition 6 (\mathbf{o} -indistinguishability and \mathbf{o} -uniformity in p). Fix $\mathbf{o} \subseteq [n]$ and $I \subseteq [n]$.

- Two tuples $x, x' \in L_I$ are \mathbf{o} -indistinguishable, written $x \approx_{\mathbf{o}} x'$, if $x \downarrow_{I \cap \mathbf{o}} = x' \downarrow_{I \cap \mathbf{o}}$.
- Two words $u = u_0 \dots u_i$ and $u' = u'_0 \dots u'_j$ over alphabet L_I are \mathbf{o} -indistinguishable, written $u \approx_{\mathbf{o}} u'$, if $i = j$ and for all $k \in \{0, \dots, i\}$ we have $u_k \approx_{\mathbf{o}} u'_k$.
- A tree t is \mathbf{o} -uniform in p iff for every pair of nodes $u, u' \in \tau$ such that $u \approx_{\mathbf{o}} u'$, we have $p \in \ell(u)$ iff $p \in \ell(u')$.

Finally, we inductively define the satisfaction relation \models for the semantics on trees, where $t = (\tau, \ell)$ is a 2^{AP} -labelled L_I -tree, u is a node and λ is a path in τ :

$$\begin{aligned}
t, u \models p &\quad \text{if } p \in \ell(u) \\
t, u \models \neg\varphi &\quad \text{if } t, u \not\models \varphi \\
t, u \models \varphi \vee \varphi' &\quad \text{if } t, u \models \varphi \text{ or } t, u \models \varphi' \\
t, u \models \mathbf{E}\psi &\quad \text{if } \exists \lambda \in \text{Paths}(u) \text{ s.t. } t, \lambda \models \psi \\
t, u \models \exists^{\mathbf{o}} p. \varphi &\quad \text{if } \exists t' \equiv_p t \text{ s.t. } t' \text{ is } \mathbf{o}\text{-uniform in } p \text{ and } \\
&\quad t', u \models \varphi. \\
t, \lambda \models \varphi &\quad \text{if } t, \lambda_0 \models \varphi \\
t, \lambda \models \neg\psi &\quad \text{if } t, \lambda \not\models \psi \\
t, \lambda \models \psi \vee \psi' &\quad \text{if } t, \lambda \models \psi \text{ or } t, \lambda \models \psi' \\
t, \lambda \models \mathbf{X}\psi &\quad \text{if } t, \lambda_{\geq 1} \models \psi \\
t, \lambda \models \psi \mathbf{U} \psi' &\quad \text{if } \exists i \geq 0 \text{ s.t. } t, \lambda_{\geq i} \models \psi' \text{ and } \\
&\quad \forall j \text{ s.t. } 0 \leq j < i, t, \lambda_{\geq j} \models \psi
\end{aligned}$$

We write $t \models \varphi$ for $t, r \models \varphi$, where r is the root of t .

Example 2. Consider the following CTL formula:

$$\text{border}(p) := \mathbf{AF}p \wedge \mathbf{AG}(p \rightarrow \mathbf{AXAG}\neg p).$$

This formula holds in a labelled tree if and only if each path contains exactly one node labelled with p . Now, consider the following $\text{QCTL}_{\text{ii}}^*$ formula:

$$\text{level}(p) := \exists^{\emptyset} p. \text{border}(p).$$

For a blind quantifier, two nodes of a tree are indistinguishable if and only if they have same depth. Therefore, this formula holds on a tree iff the p 's label all and only the nodes at some fixed depth. This formula can thus be used to capture the equal level predicate on trees. Actually, just as QCTL^* captures MSO, one can prove that $\text{QCTL}_{\text{ii}}^*$ with tree semantics subsumes MSO with equal level [17], [28], [29]. In Theorem 3

we make use of a similar observation to prove that model-checking $\text{QCTL}_{\text{ii}}^*$ is undecidable.

Model-checking problem for $\text{QCTL}_{\text{ii}}^*$ under tree semantics. For the model-checking problem, we interpret $\text{QCTL}_{\text{ii}}^*$ on unfoldings of CKSs.

Tree unfoldings $t_S(s)$. Let $\mathcal{S} = (S, R, s_i, \ell)$ be a compound Kripke structure over AP, and let $s \in S$. The *tree-unfolding of \mathcal{S} from s* is the (AP, S) -tree $t_S(s) := (\tau, \ell')$, where τ is the set of all finite paths that start in s , and for every $u \in \tau$, $\ell'(u) := \ell(\text{last}(u))$. Given a CKS \mathcal{S} , a state $s \in \mathcal{S}$ and a $\text{QCTL}_{\text{ii}}^*$ formula φ , we write $\mathcal{S}, s \models \varphi$ if $t_S(s) \models \varphi$. Write $\mathcal{S} \models \varphi$ if $t_S(s_i) \models \varphi$.

The *model-checking problem for $\text{QCTL}_{\text{ii}}^*$ under tree-semantics* is the following decision problem: given an instance (φ, \mathcal{S}) where \mathcal{S} is a CKS, and φ is a $\text{QCTL}_{\text{ii}}^*$ formula, return ‘Yes’ if $\mathcal{S} \models \varphi$ and ‘No’ otherwise.

C. Discussion of the definition of $\text{QCTL}_{\text{ii}}^*$

We now motivate in detail some aspects of $\text{QCTL}_{\text{ii}}^*$.

Modelling of imperfect information. We model imperfect information by means of local states (rather than equivalence relations) since this greatly facilitates the use of automata techniques. More precisely, in our decision procedure of Section IV, we make extensive use of an operation on tree automata called *narrowing*, which was introduced in [30] to deal with imperfect-information in the context of distributed synthesis for temporal specifications. Given an automaton \mathcal{A} that works on $X \times Y$ -trees, where X and Y are two finite sets, and assuming that we want to model an operation performed on trees while observing only the X component of each node, this narrowing operation allows one to build from \mathcal{A} an automaton \mathcal{A}' that works on X -trees, such that \mathcal{A}' accepts an X -tree if and only if \mathcal{A} accepts its widening to $X \times Y$ (see Section IV for details). One can then make this automaton \mathcal{A}' perform the desired operation, which will by necessity be performed uniformly with regards to the partial observation, since the Y component is absent from the input trees.

With our definition of compound Kripke structures, their unfoldings are trees over the Cartesian product $L_{[n]}$. To model a quantification $\exists^{\mathbf{o}} p$ with observation $\mathbf{o} \subseteq [n]$, we can thus use the narrowing operation to forget about components L_i , for $i \in [n] \setminus \mathbf{o}$. We then use the classic projection of non-deterministic tree automata to perform existential quantification on atomic proposition p . Since the choice of the p -labelling is made directly on $L_{\mathbf{o}}$ -trees, it is necessarily \mathbf{o} -uniform.

Choice of the tree semantics. QCTL^* is obtained by adding to CTL^* second-order quantification on atomic propositions. Several semantics have been considered. The two most studied ones are the *structure semantics*, in which formulas are evaluated directly on Kripke structures, and the *tree semantics*, in which Kripke structures are first unfolded into infinite trees. Tree semantics thus allows quantifiers to choose the value of a quantified atomic proposition in each *finite path* of the model, while in structure semantics the choice is only made in each state. When QCTL^* is used to express existence

of strategies, existential quantification on atomic propositions labels the structure with strategic choices; in this kind of application, structure semantics reflects so-called *positional* or *memoryless* strategies, while tree semantics captures *perfect-recall* or *memoryfull* strategies. Since in this work we are interested in perfect-recall strategies, we only consider the tree semantics.

D. Model checking QCTL_{ii}^{*}

We now prove that the model-checking problem for QCTL_{ii}^{*} under tree semantics is undecidable. This comes as no surprise since, as we will show, QCTL_{ii}^{*} can express the existence of winning strategies in imperfect-information games.

Theorem 3. *The model-checking problem for QCTL_{ii}^{*} under tree-semantics is undecidable.*

Proof. Let MSO_{eq} denote the extension of the logic MSO by a binary predicate symbol eq. Formulas of MSO_{eq} are interpreted on trees, and the semantics of eq(x, y) is that x and y have the same depth in the tree. There is a translation of MSO-formulas to QCTL^{*}-formulas that preserves satisfaction [14]. This translation can be extended to map MSO_{eq}-formulas to QCTL_{ii}^{*}-formulas using the formula level(\cdot) from Example 2 to help capture the equal-length predicate. Our result follows since the MSO_{eq}-theory of the binary tree is undecidable [17]. \square

IV. A DECIDABLE FRAGMENT OF QCTL_{ii}^{*}: HIERARCHY ON OBSERVATIONS

The main result of this section is the identification of an important decidable fragment of QCTL_{ii}^{*}.

Definition 7 (Hierarchical formulas). A QCTL_{ii}^{*} formula φ is hierarchical if for all subformulas φ_1, φ_2 of the form $\varphi_1 = \exists^{\mathbf{o}_1} p_1. \varphi'_1$ and $\varphi_2 = \exists^{\mathbf{o}_2} p_2. \varphi'_2$ where φ_2 is a subformula of φ'_1 , we have $\mathbf{o}_1 \subseteq \mathbf{o}_2$.

In other words, a formula is hierarchical if innermost propositional quantifiers observe at least as much as outermost ones. We let QCTL_{i,≤}^{*} be the set of hierarchical QCTL_{ii}^{*} formulas.

Theorem 4. *Model checking QCTL_{i,≤}^{*} under tree semantics is non-elementary decidable.*

Since our decision procedure for the hierarchical fragment of QCTL_{ii}^{*} is based on an automata-theoretic approach, we recall some definitions and results for alternating tree automata.

A. Alternating parity tree automata

We briefly recall the notion of alternating (parity) tree automata. Because it is sufficient for our needs and simplifies definitions, we assume that all input trees are complete trees. For a set Z , $\mathbb{B}^+(Z)$ is the set of formulas built from the elements of Z as atomic propositions using the connectives \vee and \wedge , and with $\top, \perp \in \mathbb{B}^+(Z)$. An *alternating tree automaton (ATA)* on (AP, X)-trees is a structure $\mathcal{A} = (Q, \delta, q_i, C)$ where Q is a finite set of states, $q_i \in Q$ is an initial state,

$\delta : Q \times 2^{\text{AP}} \rightarrow \mathbb{B}^+(X \times Q)$ is a transition function, and $C : Q \rightarrow \mathbb{N}$ is a colouring function. To ease reading we shall write atoms in $\mathbb{B}^+(X \times Q)$ between brackets, such as $[x, q]$. A *nondeterministic tree automaton (NTA)* on (AP, X)-trees is an ATA $\mathcal{A} = (Q, \delta, q_i, C)$ such that for every $q \in Q$ and $a \in 2^{\text{AP}}$, $\delta(q, a)$ is written in disjunctive normal form and for every direction $x \in X$ each disjunct contains exactly one element of $\{x\} \times Q$. Acceptance is defined as usual (see, e.g., [31]), and we let $\mathcal{L}(\mathcal{A})$ be the set of trees accepted by \mathcal{A} .

We recall three classic results on tree automata. The first one is that nondeterministic tree automata are closed under projection, and was established by Rabin to deal with second-order monadic quantification:

Theorem 5 (Projection [32]). *Given an NTA \mathcal{N} and an atomic proposition $p \in \text{AP}$, one can build an NTA $\mathcal{N} \Downarrow_p$ such that $\mathcal{L}(\mathcal{N} \Downarrow_p) = \mathcal{L}(\mathcal{N}) \Downarrow_p$.*

Because it will be important to understand the automata construction for our decision procedure in Section IV, we briefly recall that the projected automaton $\mathcal{N} \Downarrow_p$ is simply automaton \mathcal{N} with the only difference that when it reads the label of a node, it can choose whether p is there or not: if δ is the transition function of \mathcal{N} , that of $\mathcal{N} \Downarrow_p$ is $\delta'(q, a) = \delta(q, a \cup \{p\}) \vee \delta(q, a \setminus \{p\})$, for any state q and label $a \in 2^{\text{AP}}$. Another way of seeing it is that $\mathcal{N} \Downarrow_p$ first guesses a p -labelling for the input tree, and then simulates \mathcal{N} on this modified input. To prevent $\mathcal{N} \Downarrow_p$ from guessing different labels for a same node in different executions, it is crucial that \mathcal{N} be nondeterministic, which is the reason why we need the next classic result: the crucial simulation theorem, due to Muller and Schupp.

Theorem 6 (Simulation [33]). *Given an ATA \mathcal{A} , one can build an NTA \mathcal{N} such that $\mathcal{L}(\mathcal{N}) = \mathcal{L}(\mathcal{A})$.*

The third result was established by Kupferman and Vardi to deal with imperfect information aspects in distributed synthesis. To state it we need to define a widening operation on trees which simply expands the directions in a tree.

Widening [30]. Let X and Y be two finite sets, let t be an X -tree with root x , and let $y \in Y$. The Y -widening of t with root (x, y) is the $X \times Y$ -tree

$$\tau \uparrow_y^Y := \{u \in (x, y) \cdot (X \times Y)^* \mid u \downarrow_X \in \tau\}.$$

For an (AP, X)-tree $t = (\tau, \ell)$, we let $t \uparrow_y^Y := (\tau \uparrow_y^Y, \ell')$ where $\ell'(u) := \ell(u \downarrow_X)$.

Similarly to the narrowing operation, we extend this definition to tuples of local states by letting, for $J \subseteq I \subseteq [n]$, τ an L_J -tree and $z' \in L_{I \setminus J}$,

$$\tau \uparrow_z^I := \tau \uparrow_{z'}^{L_{I \setminus J}},$$

and similarly for a labelled L_J -tree t ,

$$t \uparrow_z^I := t \uparrow_{z'}^{L_{I \setminus J}}.$$

Recall that because the sets of local states L_i are disjoint, the order of local states in a tuple does not matter and we can identify L_I with $L_J \times L_{I \setminus J}$.

The rough idea of the narrowing operation on ATA is that, if one just observes X , uniform p -labellings on $X \times Y$ -trees can be obtained by choosing the labellings directly on X -trees, and then lifting them to $X \times Y$.

Theorem 7 (Narrowing [30]). *Given an ATA \mathcal{A} on $X \times Y$ -trees one can build an ATA $\mathcal{A} \downarrow_X$ on X -trees such that for all $y \in Y$, $t \in \mathcal{L}(\mathcal{A} \downarrow_X)$ iff $t \uparrow^{X \times Y}_y \in \mathcal{L}(\mathcal{A})$.*

B. Translating QCTL_{i, \subseteq} * to ATA

In order to prove Theorem 4 we need some more notations and a technical lemma that contains the automata construction.

For every $\varphi \in \text{QCTL}_{ii}^*$, we let

$$I_\varphi := \bigcap_{\mathbf{o} \in \text{Obs}(\varphi)} \mathbf{o} \subseteq [n],$$

where $\text{Obs}(\varphi)$ is the set of concrete observations that occur in φ , with the intersection over the empty set defined as $[n]$. We also let $L_\varphi := L_{I_\varphi}$ (recall that for $I \subseteq [n]$, $L_I = \prod_{i \in I} L_i$).

Our construction, that transforms a QCTL_{i, \subseteq} * formula φ and a CKS \mathcal{S} into an ATA, builds upon the classic construction from [34], which builds hesitant ATA for CTL* formulas. Since our aim here is to establish decidability and that the hesitant condition is only used to improve complexity, we do not consider it. However we need to develop an original technique to implement quantifiers with imperfect information thanks to automata narrowing and projection.

The classical approach to model checking via tree automata is to build an automaton that accepts all tree models of the input formula, and check whether it accepts the unfolding of the model [34]. We now explain how we adapt this approach.

Narrowing of non-uniform trees. Quantification on atomic propositions is classically performed by means of automata projection (see Theorem 5). But in order to obtain a labelling that is uniform with regards to the observation of the quantifier, we need to make use of the narrow operation (see Theorem 7). Intuitively, to check that a formula $\exists^\mathbf{o} p. \varphi$ holds in a tree t , we would like to work on its narrowing $t' := t \downarrow_\mathbf{o}$, guess a labelling for p on this tree thanks to automata projection, thus obtaining a tree t'_p , take its widening $t''_p := t'_p \uparrow^{[n]}$, obtaining a tree with an \mathbf{o} -uniform labelling for p , and then check that φ holds on t''_p . The problem here is that, while the narrowing $\tau \downarrow_\mathbf{o}$ of an unlabelled tree τ is well defined (see Section III-B), that of a labelled tree $t = (\tau, \ell)$ is undefined: indeed, unless t is \mathbf{o} -uniform in every atomic proposition in AP, there is no way to define the labelling of $\tau \downarrow_\mathbf{o}$ without losing information. This implies that we cannot feed (a narrowing of) the unfolding of the model to our automata. Still, we need an input tree to be successively labelled and widened to guess uniform labellings.

Splitting quantified from free propositions. To address this problem, we remark that since we are interested in model checking a QCTL_{ii}* formula φ on a CKS \mathcal{S} , the automaton that we build for φ can depend on \mathcal{S} . It can thus guess paths in \mathcal{S} , and evaluate free occurrences of atomic propositions in \mathcal{S} without reading the input tree. The input tree is thus no longer used to represent the model. However we use it to carry

labellings for quantified propositions $\text{AP}_\exists(\varphi)$: we provide the automaton with an input tree whose labelling is initially empty, and the automaton, through successive narrowing and projection operations, decorates it with uniform labellings for quantified atomic propositions.

We remark that this technique allows one to go beyond CL [19]: by separating between quantified atomic propositions (that need to be uniform) and free atomic propositions (that state facts about the model), we manage to remove the restriction present in CL, that requires that all facts about the model are known to every strategy/agent (see the Appendix).

To do this we assume without loss of generality that propositions that are quantified in φ do not appear free in φ , i.e., $\text{AP}_\exists(\varphi) \cap \text{AP}_f(\varphi) = \emptyset$. If necessary, for every $p \in \text{AP}_\exists(\varphi) \cap \text{AP}_f(\varphi)$, we take a fresh atomic proposition p' and replace all quantified occurrences of p in φ with p' . We obtain an equivalent formula φ' on $\text{AP}' := \text{AP} \cup \{p' \mid p \in \text{AP}_\exists(\varphi) \cap \text{AP}_f(\varphi)\}$ such that $\text{AP}_\exists(\varphi') \cap \text{AP}_f(\varphi') = \emptyset$. Observe also that given a formula φ such that $\text{AP}_\exists(\varphi) \cap \text{AP}_f(\varphi) = \emptyset$, a CKS \mathcal{S} and a state $s \in \mathcal{S}$, the truth value of φ in \mathcal{S}, s does not depend on the labelling of \mathcal{S} for atomic propositions in $\text{AP}_\exists(\varphi)$, which can thus be forgotten.

As a consequence, henceforth we assume that an instance (φ, \mathcal{S}) of the model-checking problem for QCTL_{ii}* is such that $\text{AP}_\exists(\varphi) \cap \text{AP}_f(\varphi) = \emptyset$, and \mathcal{S} is a CKS over $\text{AP}(\varphi)$.

Merging the decorated input tree and the model. To state the correctness of our construction, we will need to merge the labels for quantified propositions, carried by the input tree, with those for free propositions, carried by CKS \mathcal{S} . Because, through successive widenings, the input tree (represented by t in the definition below) will necessarily be a complete tree, its domain will always contain the domain of the unfolding of \mathcal{S} (represented by t' below), hence the following definition.

Definition 8 (Merge). *Let $t = (\tau, \ell)$ be a complete (AP, X) -tree and $t' = (\tau', \ell')$ an (AP', X) -tree, where $\text{AP} \cap \text{AP}' = \emptyset$. We define the merge of t and t' as the $(\text{AP} \cup \text{AP}', X)$ -tree $t \mathbin{\text{M}\kern-1.6ex\text{l}} t' := (\tau \cap \tau' = \tau', \ell'')$, where $\ell''(u) = \ell(u) \cup \ell'(u)$.*

We now describe our automata construction and establish the following lemma, which is our main technical contribution.

Lemma 1 (Translation). *Let (Φ, \mathcal{S}) be an instance of the model-checking problem for QCTL_{i, \subseteq} *. For every subformula φ of Φ and state s of \mathcal{S} , one can build an ATA \mathcal{A}_s^φ on $(\text{AP}_\exists(\Phi), L_\varphi)$ -trees such that for every $(\text{AP}_\exists(\Phi), L_\varphi)$ -tree t rooted in $s \downarrow_{I_\varphi}$,*

$$t \in \mathcal{L}(\mathcal{A}_s^\varphi) \text{ iff } t \uparrow_y^{[n]} \mathbin{\text{M}\kern-1.6ex\text{l}} t_{\mathcal{S}}(s) \models \varphi, \quad \text{where } y = s \downarrow_{[n] \setminus I_\varphi}.$$

For an L_I -tree t , from now on $t \uparrow_y^{[n]} \mathbin{\text{M}\kern-1.6ex\text{l}} t_{\mathcal{S}}(s)$ stands for $t \uparrow_y^{[n]} \mathbin{\text{M}\kern-1.6ex\text{l}} t_{\mathcal{S}}(s)$, where $y = s \downarrow_{[n] \setminus I}$: the missing local states in the root of t are filled with those from s .

Proof sketch of Lemma 1. Let (Φ, \mathcal{S}) be an instance of the model-checking problem for QCTL_{i, \subseteq} *. Let $\Phi \in \text{QCTL}_{i,\subseteq}^*$, and let $\text{AP}_\exists = \text{AP}_\exists(\Phi)$ and $\text{AP}_f = \text{AP}_f(\Phi)$, and recall that \mathcal{S} is labelled over AP_f . For each state $s \in S$ and each subformula

φ of Φ (note that all subformulas of Φ are also hierarchical), we define by induction on φ the ATA \mathcal{A}_s^φ on $(\text{AP}_\exists, L_\varphi)$ -trees.

$\varphi = p :$

We let \mathcal{A}_s^p be the ATA over $L_{[n]}$ -trees with one unique state q_ℓ , with transition function defined as follows:

$$\delta(q_\ell, a) = \begin{cases} \top & \text{if } p \in \text{AP}_f \text{ and } p \in \ell_S(s) \\ & \quad \text{or} \\ & p \in \text{AP}_\exists \text{ and } p \in a \\ & p \in \text{AP}_f \text{ and } p \notin \ell_S(s) \\ \perp & \text{if } \quad \quad \quad \text{or} \\ & p \in \text{AP}_\exists \text{ and } p \notin a \end{cases}$$

$\varphi = \neg\varphi' :$

We obtain \mathcal{A}_s^φ by dualising $\mathcal{A}_s^{\varphi'}$, a classic operation.

$\varphi = \varphi_1 \vee \varphi_2 :$

Because $I_\varphi = I_{\varphi_1} \cap I_{\varphi_2}$, and each $\mathcal{A}_s^{\varphi_i}$ works on L_{φ_i} -trees, we first narrow them so that they work on L_φ -trees: for $i \in \{1, 2\}$, we let $\mathcal{A}_i := \mathcal{A}_s^{\varphi_i} \downarrow_{I_\varphi}$. We then build \mathcal{A}_s^φ by taking the disjoint union of \mathcal{A}_1 and \mathcal{A}_2 and adding a new initial state that nondeterministically chooses which of \mathcal{A}_1 or \mathcal{A}_2 to execute on the input tree, so that $\mathcal{L}(\mathcal{A}_s^\varphi) = \mathcal{L}(\mathcal{A}_1) \cup \mathcal{L}(\mathcal{A}_2)$.

$\varphi = \mathbf{E}\psi :$

The aim is to build an automaton \mathcal{A}_s^φ that works on L_φ -trees and that on input t , checks for the existence of a path in $t \uparrow^{[n]} \wedge t_S(s)$ that satisfies ψ . Observe that a path in $t \uparrow^{[n]} \wedge t_S(s)$ is a path in $t_S(s)$, augmented with the labelling for atomic propositions in AP_\exists carried by t .

To do so, \mathcal{A}_s^φ guesses a path λ in (S, s) . It remembers the current state in S , which provides the labelling for atomic propositions in AP_f , and while it guesses λ it follows its L_φ -narrowing in its input tree t (which always exists since input to tree automata are complete trees), reading the labels to evaluate propositions in AP_\exists .

Let $\max(\psi) = \{\varphi_1, \dots, \varphi_n\}$ be the set of maximal state sub-formulas of ψ . In a first step we see these maximal state sub-formulas as atomic propositions. The formula ψ can thus be seen as an LTL formula, and we can build a nondeterministic parity word automaton $\mathcal{W}^\psi = (Q^\psi, \Delta^\psi, q_\ell^\psi, C^\psi)$ over alphabet $2^{\max(\psi)}$ that accepts exactly the models of ψ [35].³ We define the ATA \mathcal{A} that, given as input a $(\max(\psi), L_\varphi)$ -tree t , nondeterministically guesses a path λ in $t \uparrow^{[n]} \wedge t_S(s)$ and simulates \mathcal{W}^ψ on it, assuming that the labels it reads while following $\lambda \downarrow_{I_\varphi}$ in its input correctly represent the truth value of formulas in $\max(\psi)$ along λ . Recall that $S = (S, R, s_\ell, \ell_S)$; we define $\mathcal{A} := (Q, \delta, q_\ell, C)$, where

- $Q = Q^\psi \times S$,
- $q_\ell = (q_\ell^\psi, s)$,
- $C(q^\psi, s') = C^\psi(q^\psi)$, and

³Note that, as usual for nondeterministic word automata, we take the transition function of type $\Delta^\psi : Q^\psi \times 2^{\max(\psi)} \rightarrow 2^{Q^\psi}$.

- for each $(q^\psi, s') \in Q$ and $a \in 2^{\max(\psi)}$,

$$\delta((q^\psi, s'), a) = \bigvee_{q' \in \Delta^\psi(q^\psi, a)} \bigvee_{s'' \in R(s')} [s'' \downarrow_{I_\varphi}, (q', s'')].$$

The intuition is that \mathcal{A} reads the current label, chooses nondeterministically which transition to take in \mathcal{W}^ψ , chooses a next state in S and proceeds in the corresponding direction in X_φ . Thus, \mathcal{A} accepts a $\max(\varphi)$ -labelled L_φ -tree t iff there is a path in t that is the L_φ -narrowing of some path in $t_S(s)$, and that satisfies ψ , where maximal state formulas are considered as atomic propositions.

Now from \mathcal{A} we build the automaton \mathcal{A}_s^φ over L_φ -trees labelled with “real” atomic propositions in AP_\exists . In each node it visits, \mathcal{A}_s^φ guesses what should be its labelling over $\max(\psi)$, it simulates \mathcal{A} accordingly, and checks that the guess it made is correct. If the path being guessed in $t_S(s)$ is currently in node u ending with state s' , and \mathcal{A}_s^φ guesses that φ_i holds in u , it checks this guess by starting a simulation of automaton $\mathcal{A}_{s'}^{\varphi_i}$ from node $v = u \downarrow_{I_\varphi}$ in its input t .

For each $s' \in S$ and each $\varphi_i \in \max(\psi)$ we first build $\mathcal{A}_{s'}^{\varphi_i}$, and we let $\underline{\mathcal{A}}_{s'}^i := \mathcal{A}_{s'}^{\varphi_i} = (Q_{s'}^i, \delta_{s'}^i, q_{s'}^i, C_{s'}^i)$. We also let $\overline{\mathcal{A}}_{s'}^i = (\overline{Q}_{s'}^i, \overline{\delta}_{s'}^i, \overline{q}_{s'}^i, \overline{C}_{s'}^i)$ be its dualisation, and we assume w.l.o.g. that all the state sets are pairwise disjoint. Observe that because each φ_i is a maximal state sub-formula, we have $I_{\varphi_i} = I_\varphi$, so that we do not need to narrow down automata $\mathcal{A}_{s'}^i$ and $\overline{\mathcal{A}}_{s'}^i$. We define the ATA

$$\mathcal{A}_s^\varphi = (Q \cup \bigcup_{i, s'} Q_{s'}^i \cup \overline{Q_{s'}^i}, \delta', q_\ell, C'),$$

where the colours of states are left as they were in their original automaton, and δ is defined as follows. For states in $Q_{s'}^i$ (resp. $\overline{Q_{s'}^i}$), δ agrees with $\delta_{s'}^i$ (resp. $\overline{\delta}_{s'}^i$), and for $(q^\psi, s') \in Q$ and $a \in 2^{\text{AP}_\exists}$ we let $\delta'((q^\psi, s'), a)$ be the disjunction over $a' \in 2^{\max(\psi)}$ of

$$\left(\delta((q^\psi, s'), a') \wedge \bigwedge_{\varphi_i \in a'} \delta_{s'}^i(q_{s'}^i, a) \wedge \bigwedge_{\varphi_i \notin a'} \overline{\delta}_{s'}^i(\overline{q_{s'}^i}, a) \right).$$

$\varphi = \exists^0 p. \varphi' :$

We build automaton $\mathcal{A}_s^{\varphi'}$ that works on $L_{\varphi'}$ -trees; because φ is hierarchical, we have that $\mathbf{o} \subseteq I_{\varphi'}$ and we can narrow down $\mathcal{A}_s^{\varphi'}$ to work on $L_\mathbf{o}$ -trees and obtain $\mathcal{A}_1 := \mathcal{A}_s^{\varphi'} \downarrow_\mathbf{o}$. By Theorem 6 we can nondeterminise it to get \mathcal{A}_2 , which by Theorem 5 we can project with respect to p , finally obtaining $\mathcal{A}_s^\varphi := \mathcal{A}_2 \Downarrow_p$.

The proof that the construction is correct can be found in the Appendix. \square

C. Proof of Theorem 4

We can now prove Theorem 4. Let S be a CKS, $s \in S$, and $\varphi \in \text{QCTL}_{\mathbf{i} \subseteq}^*$. By Lemma 1 one can build an ATA \mathcal{A}_s^φ such that for every labelled L_φ -tree t rooted in $s \downarrow_{I_\varphi}$, it holds that $t \in \mathcal{L}(\mathcal{A}_s^\varphi)$ iff $t \uparrow^{[n]} \wedge t_S(s) \models \varphi$. Let τ be the full L_φ -tree rooted in $s \downarrow_\varphi$, and let $t = (\tau, \ell_\emptyset)$, where ℓ_\emptyset is the empty labelling. Clearly, $t \uparrow^{[n]} \wedge t_S(s) = t_S(s)$, and because t is rooted in $s \downarrow_\varphi$, we have $t \in \mathcal{L}(\mathcal{A}_s^\varphi)$ iff $t_S(s) \models \varphi$. It

only remains to build a simple deterministic tree automaton \mathcal{A} over L_φ -trees such that $\mathcal{L}(\mathcal{A}) = \{t\}$, and check for emptiness of the alternating tree automaton $\mathcal{L}(\mathcal{A} \cap \mathcal{A}_\varphi^{\mathcal{G}})$. Because nondeterminisation makes the size of the automaton gain one exponential for each nested quantifier on propositions, the procedure is nonelementary, and hardness is inherited from the model-checking problem for QCTL [14].

V. MODEL-CHECKING HIERARCHICAL INSTANCES OF SL_{ii}

In this section we establish that the model-checking problem for SL_{ii} restricted to the class of hierarchical instances is decidable (Theorem 2).

We build upon the proof in [22] that establishes the decidability of the model-checking problem for ATL_{sc}^* by reduction to the model-checking problem for QCTL^* . The main difference is that we reduce to the model-checking problem for QCTL_{ii}^* instead, using quantifiers on atomic propositions parameterised with observations that reflect the ones used in the SL_{ii} model-checking instance.

Let (Φ, \mathcal{G}) be a hierarchical instance of the SL_{ii} model-checking problem, where $\mathcal{G} = (\text{Ac}, V, E, \ell, v_i, \mathcal{O})$. We will first show how to define a CKS $\mathcal{S}_\mathcal{G}$ and a bijection $\rho \mapsto u_\rho$ between the set of finite plays ρ starting in a given position v and the set of nodes in $t_{\mathcal{S}_\mathcal{G}}(s_v)$.

Then, for every subformula φ of Φ and partial function $f : \text{Ag} \rightarrow \text{Var}$, we will define a QCTL_{ii}^* formula $(\varphi)^f$ (that will also depend on \mathcal{G}) such that the following holds:

Proposition 5. Suppose that $\text{free}(\varphi) \cap \text{Ag} \subseteq \text{dom}(f)$, and $f(a) = x$ implies $\chi(a) = \chi(x)$ for all $a \in \text{dom}(f)$. Then

$$\mathcal{G}, \chi, \rho \models \varphi \quad \text{if and only if} \quad t_{\mathcal{S}_\mathcal{G}}(s_\rho) \models (\varphi)^f.$$

Applying this to the sentence Φ , any assignment χ , and the empty function \emptyset , we get that

$$\mathcal{G}, \chi, v_i \models \Phi \quad \text{if and only if} \quad t_{\mathcal{S}_\mathcal{G}}(s_{v_i}) \models (\Phi)^\emptyset.$$

Constructing the CKS $\mathcal{S}_\mathcal{G}$. We will define $\mathcal{S}_\mathcal{G}$ so that (indistinguishable) nodes in its tree-unfolding correspond to (indistinguishable) finite plays in \mathcal{G} . The CKS will make use of atomic propositions $\text{AP}_v := \{p_v \mid v \in V\}$ (that we assume to be disjoint from AP). The idea is that p_v allows the QCTL_{ii}^* formula $(\Phi)^\emptyset$ to refer to the current position v in \mathcal{G} . Later we will see that $(\Phi)^\emptyset$ will also make use of atomic propositions $\text{AP}_c := \{p_c^x \mid c \in \text{Ac} \text{ and } x \in \text{Var}\}$ that we assume, again, are disjoint from $\text{AP} \cup \text{AP}_v$. This allows the formula to use p_c^x to refer to the actions c advised by strategies x .

Suppose $\text{Obs} = \{o_1, \dots, o_n\}$. For $i \in [n]$, define the local states $L_i := \{[v]_{o_i} \mid v \in V\}$ where $[v]_o$ is the equivalence class of v for relation \sim_o . Since we need to know the actual position of the CGS_{ii} to define the dynamics, we also let $L_{n+1} := V$.

Define the CKS $\mathcal{S}_\mathcal{G} := (S, R, s_i, \ell')$ where

- $S := \{s_v \mid v \in V\}$,
- $R := \{(s_v, s_{v'}) \mid \exists c \in \text{Ac}^{\text{Ag}} \text{ s.t. } E(v, c) = v'\} \subseteq S^2$,
- $s_i := s_{v_i}$,
- $\ell'(s_v) := \ell(v) \cup \{p_v\} \subseteq \text{AP} \cup \text{AP}_v$,

and $s_v := ([v]_{o_1}, \dots, [v]_{o_n}, v) \in \prod_{i \in [n+1]} L_i$.

We now show how to connect finite plays in \mathcal{G} with nodes in the tree unfolding of $\mathcal{S}_\mathcal{G}$. For every finite play $\rho = v_0 \dots v_k$, define the node $u_\rho := s_{v_0} \dots s_{v_k}$ in $t_{\mathcal{S}_\mathcal{G}}(s_{v_0})$ (which exists, by definition of $\mathcal{S}_\mathcal{G}$ and of tree unfoldings). Note that the mapping $\rho \mapsto u_\rho$ defines a bijection between the set of finite plays and the set of nodes in $t_{\mathcal{S}_\mathcal{G}}(s_i)$.

Constructing the $\text{QCTL}_{i,\subseteq}^*$ formulas $(\varphi)^f$. We now describe how to transform an SL_{ii} formula φ and a partial function $f : \text{Ag} \rightarrow \text{Var}$ into a QCTL_{ii}^* formula $(\varphi)^f$ (that will also depend on \mathcal{G}). Suppose that $\text{Ac} = \{c_1, \dots, c_l\}$, and define $(\varphi)^f$ by induction on φ . We begin with the simple cases: $(p)^f := p$; $(\neg\varphi)^f := \neg(\varphi)^f$; and $(\varphi_1 \vee \varphi_2)^f := (\varphi_1)^f \vee (\varphi_2)^f$.

We continue with the second-order quantifier case:

$$(\langle\langle x\rangle\rangle^o \varphi)^f := \exists \tilde{o} p_{c_1}^x \dots \exists \tilde{o} p_{c_l}^x. \varphi_{\text{str}}(x) \wedge (\varphi)^f$$

where $\tilde{o}_i := \{j \mid \mathcal{O}(o_i) \subseteq \mathcal{O}(o_j)\}$, and

$$\varphi_{\text{str}}(x) := \mathbf{AG} \bigvee_{c \in \text{Ac}} (p_c^x \wedge \bigwedge_{c' \neq c} \neg p_{c'}^x).$$

We describe this formula in words. For each possible action $c \in \text{Ac}$, an existential quantification on the atomic proposition p_c^x “chooses” for each finite play $\rho = v_0 \dots v_k$ of \mathcal{G} (or, equivalently, for each node u_ρ of the tree $t_{\mathcal{S}_\mathcal{G}}(s_{v_0})$) whether strategy x plays action c in ρ or not. Formula $\varphi_{\text{str}}(x)$ ensures that in each finite play, exactly one action is chosen for strategy x , and thus that atomic propositions p_c^x indeed characterise a strategy, call it σ_x .⁴

Moreover, a quantifier with concrete observation \tilde{o}_i receives information corresponding to observation o_i (observe that for all $i \in [n]$, $i \in \tilde{o}_i$) as well as information corresponding to coarser observations. Note that including all coarser observations does not increase the information accessible to the quantifier: indeed, one can show that two nodes are $\{i\}$ -indistinguishable if and only if they are \tilde{o}_i -indistinguishable. However, this definition of \tilde{o}_i allows us to obtain hierarchical formulas. Since quantification on propositions p_c^x is done uniformly with regards to concrete observation \tilde{o}_i , it follows that σ_x is an o_i -strategy.

Here are the remaining cases:

$$\begin{aligned} ((a, x)\varphi)^f &:= (\varphi)^{f[a \mapsto x]} \\ (\mathbf{X}\varphi_1)^f &:= \mathbf{A}(\psi_{\text{out}}(f) \rightarrow \mathbf{X}(\varphi_1)^f) \\ (\varphi_1 \mathbf{U} \varphi_2)^f &:= \mathbf{A}(\psi_{\text{out}}(f) \rightarrow (\varphi_1)^f \mathbf{U} (\varphi_2)^f) \end{aligned}$$

where

$$\psi_{\text{out}}(f) := \mathbf{G} \left(\bigwedge_{v \in V} \bigwedge_{c \in \text{Ac}^{\text{Ag}}} \left(p_v \wedge \bigwedge_{a \in \text{Ag}} p_{c_a}^{f(a)} \rightarrow \mathbf{X} p_{E(v,c)} \right) \right).$$

The formula $\psi_{\text{out}}(f)$ is used to select the unique path assuming that every player, say a , follows the strategy $\sigma_{f(a)}$.

This completes the justification of Proposition 5.

⁴More precisely, if $\varphi_{\text{str}}(x)$ holds in node u_ρ , it ensures that propositions from AP_c define a partial strategy, defined on all nodes of the subtree rooted in u_ρ . This is enough because SL_{ii} can only talk about the future: when evaluating a formula in a finite play ρ , the definition of strategies on plays that do not start with ρ is irrelevant.

Preserving hierarchy. To complete the proof we show that $(\Phi)^\emptyset$ is a hierarchical QCTL_{ii}^{*} formula. This simply follows from the fact that Φ is hierarchical in \mathcal{G} and that for every two observations o_i and o_j in Obs such that $\mathcal{O}(o_i) \subseteq \mathcal{O}(o_j)$, by definition of \tilde{o}_k we have that $\tilde{o}_i \subseteq \tilde{o}_j$.

This completes the proof of Theorem 2.

VI. OUTLOOK

We introduced SL_{ii}, a logic for reasoning about strategic behaviour in multi-player games with imperfect information. The syntax mentions observations, and thus allows one to write formulas that talk about dynamic observations. We isolated the class of hierarchical formula/model pairs Φ, \mathcal{G} and proved that one can decide if $\mathcal{G} \models \Phi$. The proof reduces (hierarchical) instances to (hierarchical) formulas of QCTL_{ii}^{*}, a low-level logic that we introduced, and that serves as a natural bridge between SL_{ii} (that talks about players and strategies) and automata constructions. We believe that QCTL_{ii}^{*} is of independent interest and deserves study in its own right.

Since one can alternate quantifiers in SL_{ii}, our decidability result goes beyond synthesis. As we showed, we can use it to decide if a game that yields hierarchical observation has a Nash equilibrium. A crude but easy analysis of our main decision procedure shows it is non-elementary.

This naturally leads to a number of avenues for future work: define and study the expressive power and computational complexity of fragments of SL_{ii} [36]; adapt the notion of hierarchical instances to allow for situations in which hierarchies can change infinitely often along a play [13]; and extend the logic to include epistemic operators for individual and common knowledge, as is done in [37], which are important for reasoning about distributed systems [21].

REFERENCES

- [1] F. Mogavero, A. Murano, G. Perelli, and M. Y. Vardi, “Reasoning about strategies: On the model-checking problem,” *TOCL*, vol. 15, no. 4, pp. 34:1–34:47, 2014.
- [2] N. Bulling and W. Jamroga, “Comparing variants of strategic ability: how uncertainty and memory influence general properties of games,” *AAMAS’14*, vol. 28, no. 3, pp. 474–518, 2014.
- [3] F. Laroussinie, N. Markey, and A. Sangnier, “ATLsc with partial observation,” in *GandALF’15*, 2015, pp. 43–57.
- [4] C. Dima and F. L. Tiplea, “Model-checking ATL under imperfect information and perfect recall semantics is undecidable,” *CoRR*, vol. abs/1102.4225, 2011.
- [5] A. Pnueli and R. Rosner, “Distributed reactive systems are hard to synthesize,” in *FOCS’90*, 1990, pp. 746–757.
- [6] G. L. Peterson and J. H. Reif, “Multiple-person alternation,” in *SFCS’79*, 1979, pp. 348–363.
- [7] O. Kupfermann and M. Y. Vardi, “Synthesizing distributed systems,” in *LICS’01*, 2001, pp. 389–398.
- [8] G. Peterson, J. Reif, and S. Azhar, “Decision algorithms for multiplayer noncooperative games of incomplete information,” *CAMWA*, vol. 43, no. 1, pp. 179–206, 2002.
- [9] B. Finkbeiner and S. Schewe, “Uniform distributed synthesis,” in *LICS’05*, 2005, pp. 321–330.
- [10] S. Pinchinat and S. Riedweg, “A decidable class of problems for control under partial observation,” *IPL*, vol. 95, no. 4, pp. 454–460, 2005.
- [11] S. Schewe and B. Finkbeiner, “Distributed synthesis for alternating-time logics,” in *ATVA’07*, 2007, pp. 268–283.
- [12] K. Chatterjee and L. Doyen, “Games with a weak adversary,” in *ICALP’14*, 2014, pp. 110–121.
- [13] D. Berwanger, A. B. Mathew, and M. Van den Bogaard, “Hierarchical information patterns and distributed strategy synthesis,” in *ATVA’15*, 2015, pp. 378–393.
- [14] F. Laroussinie and N. Markey, “Quantified CTL: expressiveness and complexity,” *LMCS*, vol. 10, no. 4, 2014.
- [15] G. Peterson, J. Reif, and S. Azhar, “Lower bounds for multiplayer noncooperative games of incomplete information,” *CAMWA*, vol. 41, no. 7, pp. 957–992, 2001.
- [16] J. Y. Halpern and M. Y. Vardi, “The complexity of reasoning about knowledge and time. i. lower bounds,” *JCSS*, vol. 38, no. 1, pp. 195–237, 1989.
- [17] H. Läuchli and C. Savioz, “Monadic second order definable relations on the binary tree,” *JSL*, vol. 52, no. 01, pp. 219–226, 1987.
- [18] P. Gastin, N. Sznajder, and M. Zeitoun, “Distributed synthesis for well-connected architectures,” *FMSD*, vol. 34, no. 3, pp. 215–237, 2009.
- [19] B. Finkbeiner and S. Schewe, “Coordination logic,” in *CSL’10*, 2010, pp. 305–319.
- [20] R. Alur, T. A. Henzinger, and O. Kupferman, “Alternating-time temporal logic,” *JACM*, vol. 49, no. 5, pp. 672–713, 2002.
- [21] R. Fagin, J. Y. Halpern, Y. Moses, and M. Y. Vardi, *Reasoning about knowledge*. MIT press Cambridge, 1995, vol. 4.
- [22] F. Laroussinie and N. Markey, “Augmenting ATL with strategy contexts,” *IC*, vol. 245, pp. 98–123, 2015.
- [23] A. Sistla, “Theoretical Issues in the Design and Verification of Distributed Systems.” Ph.D. dissertation, Harvard University, Cambridge, MA, USA, 1983.
- [24] E. A. Emerson and A. P. Sistla, “Deciding branching time logic,” in *STOC’84*, 1984, pp. 14–24.
- [25] O. Kupferman, “Augmenting branching temporal logics with existential quantification over atomic propositions,” in *CAV’95*, 1995, pp. 325–338.
- [26] O. Kupferman, P. Madhusudan, P. S. Thiagarajan, and M. Y. Vardi, “Open systems in reactive environments: Control and synthesis,” in *CONCUR’00*, 2000, pp. 92–107.
- [27] T. French, “Decidability of quantified propositional branching time logics,” in *AJCAI’01*, 2001, pp. 165–176.
- [28] C. C. Elgot and M. O. Rabin, “Decidability and undecidability of extensions of second (first) order theory of (generalized) successor,” *JSL*, vol. 31, no. 2, pp. 169–181, 1966.
- [29] W. Thomas, “Infinite trees and automaton-definable relations over omega-words,” *TCS*, vol. 103, no. 1, pp. 143–159, 1992.
- [30] O. Kupferman and M. Y. Vardi, “Church’s problem revisited,” *BSL*, pp. 245–263, 1999.
- [31] D. E. Muller and P. E. Schupp, “Alternating automata on infinite trees,” *TCS*, vol. 54, pp. 267–276, 1987.
- [32] M. O. Rabin, “Decidability of second-order theories and automata on infinite trees,” *TAMS*, vol. 141, pp. 1–35, 1969.
- [33] D. E. Muller and P. E. Schupp, “Simulating alternating tree automata by nondeterministic automata: New results and new proofs of the theorems of Rabin, McNaughton and Safra,” *TCS*, vol. 141, no. 1&2, pp. 69–107, 1995.
- [34] O. Kupferman, M. Y. Vardi, and P. Wolper, “An automata-theoretic approach to branching-time model checking,” *JACM*, vol. 47, no. 2, pp. 312–360, 2000.
- [35] M. Y. Vardi and P. Wolper, “Reasoning about infinite computations,” *IC*, vol. 115, no. 1, pp. 1–37, 1994.
- [36] F. Mogavero, A. Murano, G. Perelli, and M. Y. Vardi, “What makes ATL* decidable? a decidable fragment of Strategy Logic,” in *CONCUR’12*, 2012, pp. 193–208.
- [37] P. Cermák, A. Lomuscio, F. Mogavero, and A. Murano, “MCMAS-SLK: A model checker for the verification of strategy logic specifications,” in *CAV’14*, 2014, pp. 525–532.
- [38] W. Zielonka, “Infinite games on finitely coloured graphs with applications to automata on infinite trees,” *TCS*, vol. 200, no. 1-2, pp. 135–183, 1998.

APPENDIX

A. Strategy logic under imperfect information

We establish two facts about SL_{ii} mentioned in the body: 1) the statement $\mathcal{G}, \chi, \rho \models \varphi$ is independent of χ if φ is a sentence; 2) Proposition 1 which says that the perfect-information fragment of SL_{ii} is a notational variant of SL given in [1].

Truth of sentences are independent of the assignment.

We begin with a formal definition of free players and variables in SL_{ii} .

Definition 9 (Free players and variables). *The set $\text{free}(\varphi)$ of free players and free variables of an SL_{ii} formula φ is defined as follows:*

- $\text{free}(p) := \emptyset$, where $p \in \text{AP}$.
- $\text{free}(\neg\varphi) := \text{free}(\varphi)$.
- $\text{free}(\varphi_1 \vee \varphi_2) := \text{free}(\varphi_1) \cup \text{free}(\varphi_2)$.
- $\text{free}(\mathbf{X}\varphi) := \text{Ag} \cup \text{free}(\varphi)$.
- $\text{free}(\varphi_1 \mathbf{U} \varphi_2) := \text{Ag} \cup \text{free}(\varphi_1) \cup \text{free}(\varphi_2)$.
- $\text{free}(\langle x \rangle^o \varphi) := \text{free}(\varphi) \setminus \{x\}$.
- $\text{free}((a, x)\varphi) := \text{free}(\varphi)$, if $a \notin \text{free}(\varphi)$.
- $\text{free}((a, x)\varphi) := (\text{free}(\varphi) \setminus \{a\}) \cup \{x\}$, if $a \in \text{free}(\varphi)$.

Lemma 2. *For all CGS_{ii} \mathcal{G} , finite plays ρ , assignments χ, χ' , and SL_{ii} sentences Φ ,*

$$\mathcal{G}, \chi, \rho \models \Phi \quad \text{iff} \quad \mathcal{G}, \chi', \rho \models \Phi.$$

Proof. Since a sentence has no free variables or players, it is enough to prove the following for formulas φ : if χ and χ' agree on the free variables and players of φ , then $\mathcal{G}, \chi, \rho \models \varphi$ iff $\mathcal{G}, \chi', \rho \models \varphi$. The proof is by induction on φ .

The case of an atom is immediate since the semantics do not depend on the assignment.

The case of negation follows immediately from the inductive hypothesis using the fact that $\text{free}(\neg\varphi) = \text{free}(\varphi)$. The case of disjunction is similar.

For the quantifier case $\langle x \rangle^o \varphi$ use the fact that $\chi[x \mapsto \sigma]$ and $\chi'[x \mapsto \sigma]$ agree on the free placeholders of φ (since we assumed that χ, χ' agree on the free placeholders of $\langle x \rangle^o \varphi$).

For the binding case $(a, x)\varphi$ use the fact that $\chi[a \mapsto \chi(x)]$ and $\chi'[a \mapsto \chi'(x)]$ agree on the free placeholders of φ .

For the next-operator case $\mathbf{X}\varphi$ use the fact that $\text{out}(\chi, \rho) = \text{out}(\chi', \rho)$ (since out only depends on the strategies assigned to players), and χ, χ' agree on $\text{free}(\varphi)$ since $\text{free}(\varphi) \subseteq \text{free}(\mathbf{X}\varphi)$. The case of \mathbf{U} is similar. \square

1) *Comparison of the perfect-information fragment of SL_{ii} with the definition of SL from [1]:* We recall the definitions of the syntax and semantics of SL .

Definition 10 (SL syntax [1]). *The syntax of SL is defined by the following grammar:*

$$\varphi := p \mid \neg\varphi \mid \varphi \vee \varphi \mid \mathbf{X}\varphi \mid \varphi \mathbf{U} \varphi \mid \langle x \rangle \varphi \mid (a, x)\varphi$$

where $p \in \text{AP}$, $x \in \text{Var}$, and $a \in \text{Ag}$.

An *assignment* is a partial function $\chi : \text{Ag} \cup \text{Var} \rightarrow \text{Str}$, assigning to each player and variable in its domain a strategy.

For an assignment χ , a player a and a strategy σ , $\chi[a \mapsto \sigma]$ is the assignment of domain $\text{dom}(\chi) \cup \{a\}$ that maps a to σ and is equal to χ on the rest of its domain, and $\chi[x \mapsto \sigma]$ is defined similarly, where x is a variable. An assignment χ is *player-complete* for \mathcal{G} , or simply *player-complete* when \mathcal{G} is clear from the context, if it assigns a strategy to each player in \mathcal{G} , i.e., $\text{Ag} \subseteq \text{dom}(\chi)$. For a player-complete assignment χ and a position v , let $\text{out}(\chi, v)$ be the infinite play obtained when the game starts in v and all players follow the strategies assigned by χ : $\text{out}(\chi, v) := v_0 v_1 \dots$ with $v_0 = v$ and for each $i \geq 0$, $v_{i+1} = E(v_i, c)$, where $c = (\chi(a)(v_0 \dots v_i))_{a \in \text{Ag}}$.

In addition, given a formula $\varphi \in \text{SL}$, an assignment is *variable-complete* for φ , or simply *variable-complete* when φ is clear from the context, if $\text{free}(\varphi) \cap \text{Var} \subseteq \text{dom}(\chi)$.

Given an assignment χ and an initial play ρ , define the ρ -translation of χ as the assignment χ^ρ such that $\text{dom}(\chi^\rho) = \text{dom}(\chi)$, and for every $l \in \text{dom}(\chi^\rho)$, $\chi^\rho(l) = \chi(l)^\rho$. We also define the *global translation* of a complete assignment χ together with a position v as follows: for every $i \in \mathbb{N}$, the i -*global translation* of (χ, v) is defined as $(\chi, v)^i := (\chi^{\pi \leq i}, \pi_i)$, where $\pi = \text{out}(\chi, v)$.

Models of SL formulas are concurrent games structures (CGS), i.e., tuples $\mathcal{G} = (\text{Ac}, V, E, \ell, v_\ell)$ where the entries are as for CGS_{ii} (but without Obs and \mathcal{O}).

Definition 11 (SL Semantics [1]). *Let φ be an SL -formula. The semantics of φ in a CGS \mathcal{G} at position $v \in \mathcal{G}$ with a variable-complete assignment χ is defined inductively as follows:*

$$\begin{aligned} \mathcal{G}, \chi, v \models p &\quad \text{if } p \in \ell(v) \\ \mathcal{G}, \chi, v \models \neg\varphi &\quad \text{if } \mathcal{G}, \chi, v \not\models \varphi \\ \mathcal{G}, \chi, v \models \varphi \vee \varphi' &\quad \text{if } \mathcal{G}, \chi, v \models \varphi \text{ or } \mathcal{G}, \chi, v \models \varphi' \\ \mathcal{G}, \chi, v \models \langle x \rangle \varphi &\quad \text{if } \exists \sigma \in \text{Str} \text{ s.t. } \mathcal{G}, \chi[x \mapsto \sigma], v \models \varphi \\ \mathcal{G}, \chi, v \models (a, x)\varphi &\quad \text{if } \mathcal{G}, \chi[a \mapsto \chi(x)], v \models \varphi \end{aligned}$$

If, in addition, χ is player-complete, then

$$\begin{aligned} \mathcal{G}, \chi, v \models \mathbf{X}\varphi &\quad \text{if } \mathcal{G}, (\chi, v)^1 \models \varphi \\ \mathcal{G}, \chi, v \models \varphi \mathbf{U} \varphi' &\quad \text{if } \exists i \geq 0 \text{ s.t. } \mathcal{G}, (\chi, v)^i \models \varphi' \text{ and} \\ &\quad \forall j \text{ s.t. } 0 \leq j < i, \mathcal{G}, (\chi, v)^j \models \varphi \end{aligned}$$

For a sentence $\varphi \in \text{SL}$ define $\mathcal{G}, v \models \varphi$ if $\mathcal{G}, \emptyset, v \models \varphi$ (where \emptyset is the empty assignment), and write $\mathcal{G} \models \varphi$ if $\mathcal{G}, v_\ell \models \varphi$. This completes the recap of the syntax and semantics of SL .

Proof of proposition 1. We prove this in two steps. First, note that in the definition of SL , one can restrict to complete assignments when defining \models . Indeed, an easy induction shows that for all partial assignments χ, χ' such that χ' extends χ (i.e., $\text{dom}(\chi) \subseteq \text{dom}(\chi')$ and $\chi(l) = \chi'(l)$ for all $l \in \text{dom}(\chi)$), we have that $\mathcal{G}, \chi, v \models \varphi$ iff $\mathcal{G}, \chi', v \models \varphi$. Thus, from now on we assume that all assignments of SL are complete. In addition, in [1] the authors define $\mathcal{G}, v \models \varphi$, for φ a sentence, as $\mathcal{G}, \emptyset, v \models \varphi$ where \emptyset is the empty assignment. This is equivalent to stating that $\mathcal{G}, \chi, v \models \varphi$ for all complete assignments χ .

To prove proposition 1, let φ be an SL formula. Fix an observation symbol o and let $\text{Obs} := \{o\}$. Define the SL_{ii}

formula φ' by annotating every strategy quantifier in φ by o , i.e., by replacing $\langle\langle x\rangle\rangle$ by $\langle\langle x\rangle\rangle^o$. Let \mathcal{G} be a CGS. Define the CGS_{ii} \mathcal{G}' to be $(\mathcal{G}, \mathcal{O}_{\text{bs}}, \mathcal{O})$ where $\mathcal{O}(o) = \{(v, v) : v \in V\}$ is the identity observation. We now prove that for every complete assignment χ , and $v \in \mathcal{G}$, $\mathcal{G}, \chi, v \models \varphi$ iff $\mathcal{G}', \chi, v \models \varphi'$, which establishes the proposition when φ is a sentence. This follows by induction on φ using the following inductive hypothesis: For all $n \in \mathbb{N}$,

$$\mathcal{G}, (\chi, v)^n \models \varphi \text{ iff } \mathcal{G}', \chi, \pi^{\leq n} \models \varphi,$$

where $\pi = \text{out}(\chi, v)$. The inductive step is immediate from the definitions of the semantics of \mathbf{SL} and \mathbf{SL}_{ii} .

B. Translation of CL into \mathbf{SL}_{ii}

In this section we prove Proposition 4 by giving two canonical translations: first from CL-instances into QCTL_{i,≤}^{*}-instances, and then translating QCTL_{ii}^{*}-instances into hierarchical \mathbf{SL}_{ii} -instances.

We briefly recall the syntax and semantics of CL, and refer to [19] for further details. For definitions about trees, see Section III-B.

Notation for trees. Note that our definition for trees slightly differs from the one in [19], where the root is the empty word. Here we adopt this convention to keep closer to notations in [19]. Thus, (Y, X) -trees in CL are of the form (τ, l) where $\tau \subseteq X^*$ and $l : \tau \rightarrow 2^Y$.

For two disjoint sets X and Y , we identify $2^X \times 2^Y$ with $2^{X \cup Y}$. Let X and Y be two sets with $Z = X \cup Y$, and let M and N be two disjoint sets. Given an M -labelled 2^Z -tree $t = (\tau, \ell_M)$ and an N -labelled 2^Z -tree $t' = (\tau', \ell_N)$ with same domain $\tau = \tau'$, we define $t \uplus t' := (\tau, \ell')$, where for every $u \in \tau$, $\ell'(u) = \ell_M(u) \cup \ell_N(u)$. Now, given a complete M -labelled 2^X -tree $t = ((2^X)^*, \ell_M)$ and a complete N -labelled 2^Y -tree $t' = ((2^Y)^*, \ell_N)$, we define $t \oplus t' := t \uplus_{2^{Z \setminus X}} t' \uplus_{2^{Z \setminus Y}}$.

CL Syntax. Let \mathcal{C} be a set of *coordination variables*, and let \mathcal{S} be a set of *strategy variables* disjoint from \mathcal{C} . The syntax of CL is given by the following grammar:

$$\varphi ::= x \mid \neg\varphi \mid \varphi \vee \varphi \mid \mathbf{X}\varphi \mid \varphi \mathbf{U}\varphi \mid \exists C \exists s. \varphi$$

where $x \in \mathcal{C} \cup \mathcal{S}$, $C \subseteq \mathcal{C}$ and $s \in \mathcal{S}$, and with the restriction that each coordination variable appears in at most once *subtree quantifier* $\exists C \exists s.$, and similarly for strategy variables.

The notion of free and bound (coordination or strategy) variables is as usual. The set of free coordination variables in φ is noted \mathcal{F}_φ . A bound coordination variable c is *visible* to a strategy variable s if s is in the scope of the quantifier that introduces c , and $\text{Scope}_\varphi(s)$ is the union of the set of bound coordination variables visible to s and the set of free coordination variables (note that this union is disjoint). We will see, in the semantics, that the meaning of a bound strategy variable s is a strategy $f_s : (2^{\text{Scope}_\varphi(s)})^* \rightarrow 2^{\{s\}}$. Free strategy variables are called *atomic propositions*, and we denote the set of atomic propositions in φ by AP_φ .

CL Semantics. A CL formula φ is evaluated on a complete AP_φ -labelled $2^{\mathcal{F}_\varphi}$ -tree t . An $(\text{AP}_\varphi, 2^{\mathcal{F}_\varphi})$ -tree $t = (\tau, \ell)$ satisfies a CL formula φ if for every path λ that starts in the

root we have $t, \lambda, 0 \models \varphi$, where the satisfaction of a formula at position $i \geq 0$ of a path λ is defined inductively as follows:

$$\begin{aligned} t, \lambda, i \models p & \quad \text{if } p \in \ell(\lambda_i) \\ t, \lambda, i \models \neg\varphi' & \quad \text{if } t, \lambda, i \not\models \varphi' \\ t, \lambda, i \models \varphi_1 \vee \varphi_2 & \quad \text{if } t, \lambda, i \models \varphi_1 \text{ or } t, \lambda, i \models \varphi_2 \\ t, \lambda, i \models \mathbf{X}\varphi' & \quad \text{if } t, \lambda, i + 1 \models \varphi' \\ t, \lambda, i \models \varphi_1 \mathbf{U}\varphi_2 & \quad \text{if } \exists j \geq i \text{ s.t. } t, \lambda, j \models \varphi_2 \text{ and} \\ & \quad \forall k \text{ s.t. } i \leq k < j, t, \lambda, k \models \varphi_1 \\ t, \lambda, i \models \exists C \exists s. \varphi' & \quad \text{if } \exists f : (2^{\text{Scope}_\varphi(s)})^* \rightarrow 2^{\{s\}} \text{ s.t.} \\ & \quad t_{\lambda_i} \oplus ((2^{\text{Scope}_\varphi(s)})^*, f) \models \varphi', \end{aligned}$$

where t_{λ_i} is the subtree of t rooted in λ_i .

First, observe that in the last inductive case, t_{λ_i} being a $2^{\mathcal{F}_\varphi}$ -tree, $t_{\lambda_i} \oplus ((2^{\text{Scope}_\varphi(s)})^*, f)$ is a $2^{\mathcal{F}_\varphi \cup \text{Scope}_\varphi(s)}$ -tree. By definition, $\text{Scope}_\varphi(s) = \mathcal{F}_\varphi \cup C = \mathcal{F}_{\varphi'}$. It follows that $\mathcal{F}_\varphi \cup \text{Scope}_\varphi(s) = \text{Scope}_\varphi(s) = \mathcal{F}_{\varphi'}$, hence φ' is indeed evaluated on a $\mathcal{F}_{\varphi'}$ -tree.

Remark 1. Note that all strategies observe the value of all atomic propositions. Formally, for every CL-formula φ of the form $\varphi = \exists C_1 \exists s_1. \dots, \exists C_i \exists s_i. \varphi'$ evaluated on a $2^{\mathcal{F}_\varphi}$ -tree $t = (\tau, \ell)$, φ' is evaluated on a $2^{\mathcal{F}_\varphi \cup C_1 \cup \dots \cup C_i}$ -tree $t' = (\tau', \ell')$ such that for every $p \in \text{AP}_\varphi$, for every pair of nodes $u, u' \in t'$ such that $u \downarrow_{2^{\mathcal{F}_\varphi}} = u' \downarrow_{2^{\mathcal{F}_\varphi}}$, it holds that $p \in \ell'(u)$ iff $p \in \ell'(u')$.

Thus, in CL one cannot directly capture strategic problems on concurrent game structures where atomic propositions are not observable to all players. This contrasts with the definition of ATL with imperfect information in [20, Section 7.1] where imperfect information of agents is modelled by defining which atomic propositions they can observe.

We now describe a natural translation of CL-instances to \mathbf{SL}_{ii} -instances. This translation has two consequences:

- 1) It reduces the model-checking problem of CL to that of the hierarchical fragment of \mathbf{SL}_{ii} .
- 2) It shows that CL only produces instances in which all atoms are uniform with regard to all observations, i.e., instances (Φ, \mathcal{G}) such that for every $p \in \text{AP}$ and $o \in \text{Obs}$, $v \sim_o v'$ implies $p \in \ell(v) \leftrightarrow p \in \ell(v')$.

The input to the model-checking problem for CL consists of a CL formula φ and a finite representation of a $(\text{AP}_\varphi, 2^{\mathcal{F}_\varphi})$ -tree t . The standard assumption is to assume t is a regular tree, i.e., is the unfolding of a finite structure. Precisely, a *finite representation* of a $(\text{AP}_\varphi, 2^{\mathcal{F}_\varphi})$ -tree $t = (\tau, \ell)$ is a structure $\mathcal{S} = (S, R, s_\ell, \ell)$ such that

- $S = 2^{\mathcal{F}_\varphi}$,
- $s_\ell \in S$,
- $R = S \times S$,
- $\ell : S \rightarrow 2^{\text{AP}_\varphi}$,

and $t = t_{\mathcal{S}}(s_\ell)$ is the unfolding of \mathcal{S} .

Thus, an *instance* of the model-checking problem for CL is a pair (φ, \mathcal{S}) where φ is a CL formula (over variables

$\mathcal{S} \cup \mathcal{C}$), and $\mathcal{S} = (S, R, s_\ell, \ell)$ is a finite representation of a $(\text{AP}_\varphi, 2^{\mathcal{F}_\varphi})$ -tree. The *model-checking problem for CL* is the following decision problem: given an instance (φ, \mathcal{S}) , return ‘Yes’ if $t_{\mathcal{S}}(s_\ell) \models \varphi$ and ‘No’ otherwise.

We will present the translation in two steps: first from CL-instances into $\text{QCTL}_{i,\subseteq}^*$ -instances, and then from QCTL_{ii}^* -instances to SL_{ii} -instances such that $\text{QCTL}_{i,\subseteq}^*$ -instances translate to hierarchical SL_{ii} -instances. We remind the reader that we use tree-semantics for $\text{QCTL}_{i,\subseteq}^*$ (indeed, we only defined tree-semantics, not structure-semantics, for $\text{QCTL}_{i,\subseteq}^*$).

1) *Translating CL to $\text{QCTL}_{i,\subseteq}^*$:* Let (φ, \mathcal{S}) be an instance of the model-checking problem for CL, where $\mathcal{S} = (S, R, s_\ell, \ell)$. We will construct a $\text{QCTL}_{i,\subseteq}^*$ -instance $(\bar{\varphi}, \bar{\mathcal{S}})$ such that $\mathcal{S} \models \varphi$ iff $\bar{\mathcal{S}} \models \bar{\varphi}$. Let $\overline{\text{AP}}$ be the set of all strategy variables occurring in φ , let $\mathcal{C}(\varphi)$ be the set of coordination variables that appear in φ , and assume, w.l.o.g., that $\mathcal{C}(\varphi) = [n]$ for some $n \in \mathbb{N}$. Let $\text{hidden}(\varphi) := \mathcal{C}(\varphi) \setminus \mathcal{F}_\varphi$.

First, we define the CKS $\bar{\mathcal{S}}$ over $\overline{\text{AP}}$: the idea is to add in the structure \mathcal{S} the local states corresponding to coordination variables that are not seen by all the strategies.

Formally, $\bar{\mathcal{S}} := (\bar{S}, \bar{R}, \bar{s}_\ell, \bar{\ell})$ where

- $\bar{S} = \prod_{c \in \mathcal{C}(\varphi)} L_c$ where $L_c = \{c_0, c_1\}$ (for all $c \in \mathcal{F}_\varphi$),
- $\bar{R} = \bar{S} \times \bar{S}$,
- $\bar{s}_\ell \in \bar{S}$ is any state s such that $s \downarrow_{\mathcal{F}_\varphi} = s_\ell$, and
- for every $s \in \bar{S}$, $\bar{\ell}(s) = \ell(s \downarrow_{\mathcal{F}_\varphi})$.

Second, we define concrete observations corresponding to strategy variables in φ . As explained in [19], and as reflected in the semantics of CL, the intuition is that a strategy variable s in formula φ observes coordination variables $\text{Scope}_\varphi(s)$. Therefore, we simply define, for each strategy variable s in φ , the concrete observation $\mathbf{o}_s := \text{Scope}_\varphi(s)$.

Finally, we define the QCTL_{ii}^* formula $\bar{\varphi}$. This is done by induction on φ as follows (recall that we take for atomic propositions in QCTL_{ii}^* the set of all strategy variables in φ):

$$\begin{aligned} \bar{x} &:= x \\ \bar{\neg\varphi} &:= \neg\bar{\varphi} \\ \bar{\varphi_1 \vee \varphi_2} &:= \bar{\varphi_1} \vee \bar{\varphi_2} \\ \bar{\mathbf{X}\varphi} &:= \mathbf{X}\bar{\varphi} \\ \bar{\varphi_1 \mathbf{U} \varphi_2} &:= \bar{\varphi_1} \mathbf{U} \bar{\varphi_2} \\ \bar{\exists C \exists s. \varphi} &:= \exists^{\mathbf{o}_s} s. \mathbf{A}\bar{\varphi} \end{aligned}$$

In the last case, note that $C \subseteq \mathbf{o}_s = \text{Scope}_\varphi(s)$.

Note that $\bar{\varphi}$ is a hierarchical QCTL_{ii}^* -formula. Also, one can easily check that the following holds:

Lemma 3. $t_{\mathcal{S}}(s_\ell) \models \varphi$ iff $t_{\bar{\mathcal{S}}}(\bar{s}_\ell) \models \mathbf{A}\bar{\varphi}$.

Importantly, we notice that $\mathbf{A}\bar{\varphi} \in \text{QCTL}_{i,\subseteq}^*$, and that:

Lemma 4. For every $x \in \text{AP}_\varphi$ and every s quantified in φ , $t_{\bar{\mathcal{S}}}(\bar{s}_\ell)$ is \mathbf{o}_s -uniform in x .

2) *Translation from QCTL_{ii}^* to SL_{ii} :* We now present a translation of QCTL_{ii}^* -instances to SL_{ii} -instances. It is a simple adaption of the the reduction from the model-checking

problem for QCTL^* to the model-checking problem for ATL_{sc}^* presented in [22].

Let $(\mathcal{S} = (S, R, s_\ell, \ell), \varphi)$ be an instance of the model-checking problem for QCTL_{ii}^* , where the set of states is $S \subseteq \prod_{i \in [n]} L_i$. We assume, without loss of generality, that every atomic proposition is quantified at most once, and that if it appears quantified it does not appear free. Also, let $\text{AP}_\exists(\varphi) = \{p_1, \dots, p_k\}$ be the set of atomic propositions quantified in φ , and for $i \in [k]$, let \mathbf{o}_i be the concrete observation associated to the quantifier on p_i .

We build the CGS_{ii} $\mathcal{G}_\mathcal{S} := (\text{Ac}, V, E, \ell', v_\ell, \mathcal{O})$ over agents $\text{Ag} := \{a_0, a_1, \dots, a_k\}$, observations $\text{Obs} := \{o_0, o_1, \dots, o_k\}$ and atomic propositions $\text{AP} := \text{AP}(\varphi) \cup \{p_S\}$, where p_S is a fresh atomic proposition. Intuitively, agent a_0 is in charge of choosing transitions in \mathcal{S} , while agent a_i for $i \geq 1$ is in charge of choosing the valuation for $p_i \in \text{AP}_\exists(\varphi)$.

To this aim, we let

$$V := \begin{cases} \{v_s \mid s \in S\} \cup \\ \{v_{s,i} \mid s \in S \text{ and } i \in [k]\} \cup \\ \{v_{p_i} \mid 0 \leq i \leq k\} \cup \\ \{v_\perp\} \end{cases}$$

and

$$\text{Ac} := \{c^s \mid s \in S\} \cup \{c^i \mid 0 \leq i \leq k\}.$$

In positions of the form v_s with $s \in S$, transitions are determined by the action chosen by agent a_0 . First, she can choose to simulate a transition in \mathcal{S} : for every joint action $c \in \text{Ac}^{\text{Ag}}$ such that $c_0 = c^{s'}$,

$$E(v_s, c) := \begin{cases} v_{s'} & \text{if } R(s, s') \\ v_\perp & \text{otherwise.} \end{cases}$$

She can also choose to move to a position in which agent a_i will choose the valuation for p_i in the current node: for every joint action $c \in \text{Ac}^{\text{Ag}}$ such that $c_0 = c^i$,

$$E(v_s, c) := \begin{cases} v_{s,i} & \text{if } i \neq 0 \\ v_\perp & \text{otherwise.} \end{cases}$$

Next, in a position of the form $v_{s,i}$, agent a_i determines the transition, which codes the labelling of p_i in the current node: choosing c^i means that p_i holds in the current node, choosing any other action codes that p_i does not hold. Formally, for every joint action $c \in \text{Ac}^{\text{Ag}}$,

$$E(v_{s,i}, c) := \begin{cases} v_{p_i} & \text{if } c_i = c^i \\ v_\perp & \text{otherwise.} \end{cases}$$

Positions of the form $v_{s,i}$ and v_\perp are sink positions.

The labelling function ℓ' is defined as follows:

$$\ell'(v) := \begin{cases} \ell(s) \cup \{p_S\} & \text{if } v \in \{v_s \mid s \in S\} \\ \emptyset & \text{if } v \in \{v_{s,i} \mid s \in S, i \in [k]\} \\ \{p_i\} & \text{if } v = v_{p_i} \text{ with } i \in [k] \end{cases}$$

Finally we let $v_\iota := v_{s_\iota}$ and we define the observation interpretation as follows:

$$\mathcal{O}(o_0) := \{(v, v) \mid v \in V\},$$

meaning that agent a_0 has perfect information, and for $i \in [k]$, $\mathcal{O}(o_i)$ is the smallest reflexive relation such that

$$\mathcal{O}(o_i) \supseteq \bigcup_{s, s' \in S} \{(v_s, v_{s'}), (v_{s,i}, v_{s',i}) \mid s \approx_{\mathbf{o}_i} s'\}.$$

We explain the latter definition. First, observe that for every finite play ρ in \mathcal{G}_S that stays in $V_S = \{v_s \mid s \in S\}$, writing $\rho = v_{s_0} \dots v_{s_n}$, one can associate a finite path $\lambda_\rho = s_0 \dots s_n$ in S . This mapping actually defines a bijection between the set of finite paths in S that start in s_ι and the set of finite plays in \mathcal{G}_S that remain in V_S .

Now, according to the definition of the transition function, a strategy σ_i for agent i with $i \in [k]$ is only relevant on finite plays of the form $\rho = \rho' \cdot v_{s,i}$, where $\rho' \in V_S^*$, and $\sigma_i(\rho)$ is meant to determine whether p_i holds in $\lambda_{\rho'}$. If σ_i is \mathbf{o}_i -uniform, by definition of $\mathcal{O}(o_i)$, it determines an \mathbf{o}_i -uniform labelling for p in $t_S(s_\iota)$. Reciprocally, an \mathbf{o}_i -uniform labelling for p in $t_S(s_\iota)$ induces an $\mathcal{O}(o_i)$ -strategy for agent a_i .

It remains to transform φ into an SL_{ii} -formula. To simulate the path quantifier of QCTL_{ii}^* , a strategy for agent a_0 is enough as she alone chooses the path in S . However the semantics of SL_{ii} is only defined on complete assignments, and we therefore need to make sure that all agents are bound to a strategy before using a temporal operator, and we must do so without breaking the hierarchy. For this we record in the translation what was the observation of the last propositional quantifier translated.

We define the SL_{ii} formula $\overline{\varphi}^{\mathbf{o}_i}$ by induction on φ as follows:

$$\begin{aligned} \overline{p}^{\mathbf{o}_i} &:= \begin{cases} \mathbf{A}^{\mathbf{o}_i} \langle\langle x_0 \rangle\rangle^{o_0}(a_0, x_0)(a_i, x_i) \mathbf{X} \mathbf{X} p & \text{if } p = p_i \\ p & \text{otherwise} \end{cases} \\ \overline{\neg\varphi}^{\mathbf{o}_i} &:= \neg\overline{\varphi}^{\mathbf{o}_i} \\ \overline{\varphi_1 \vee \varphi_2}^{\mathbf{o}_i} &:= \overline{\varphi_1}^{\mathbf{o}_i} \vee \overline{\varphi_2}^{\mathbf{o}_i} \\ \overline{\mathbf{X}\varphi}^{\mathbf{o}_i} &:= \mathbf{X}\overline{\varphi}^{\mathbf{o}_i} \\ \overline{\varphi_1 \mathbf{U} \varphi_2}^{\mathbf{o}_i} &:= \overline{\varphi_1}^{\mathbf{o}_i} \mathbf{U} \overline{\varphi_2}^{\mathbf{o}_i} \\ \overline{\mathbf{E}\psi}^{\mathbf{o}_i} &:= \mathbf{A}^{\mathbf{o}_i} \langle\langle x_0 \rangle\rangle^{o_0}(a_0, x_0)(\mathbf{G} p_S \wedge \overline{\psi}^{\mathbf{o}_i}) \\ \overline{\exists^{o_j} p_j. \varphi}^{\mathbf{o}_i} &:= \langle\langle x_j \rangle\rangle^{o_j} \overline{\psi}^{\mathbf{o}_j} \end{aligned}$$

where $\mathbf{A}^{\mathbf{o}_i}$ is a macro for $\llbracket x_0 \rrbracket^{o_i} \dots \llbracket x_k \rrbracket^{o_i}(a_0, x_0) \dots (a_k, x_k)$. The cases for path formulas are similar. The universal quantification on paths is just used to obtain a complete assignment, then we redefine the relevant strategies.

We have the following:

Lemma 5. $\mathcal{S} \models \varphi$ iff $\mathcal{G}_S \models \overline{\varphi}^{[n]}$.

We observe that if φ is hierarchical, then $(\overline{\varphi}^{[n]}, \mathcal{G}_S)$ is a hierarchical instance, and:

Lemma 6. For every $p \in \text{AP}_f(\varphi)$ and for every $i \in [k]$, if $t_S(s_\iota)$ is \mathbf{o}_i -uniform in p then $v \sim_{\mathbf{o}_i} v'$ implies that $p \in \ell(v)$ iff $p \in \ell(v')$.

C. Proof of Theorem 3

Theorem 3. The model-checking problem for QCTL_{ii}^* under tree-semantics is undecidable.

Proof. Let MSO_{eq} denote the extension of the logic MSO (without unary predicates) by a binary predicate symbol eq . MSO_{eq} is interpreted on the full binary tree, and the semantics of $\text{eq}(x, y)$ is that x and y have the same depth in the tree. We show how to effectively translate MSO_{eq} into QCTL_{ii}^* ; then our result follows since the MSO_{eq} -theory of the binary tree is undecidable [17].

The translation of MSO to QCTL^* from [14] can be extended to one from MSO_{eq} to QCTL_{ii}^* by adding rules for the equal level predicate. Indeed, for $\varphi(x, x_1, \dots, x_i, X_1, \dots, X_j) \in \text{MSO}$, we inductively define the QCTL_{ii}^* formula $\widehat{\varphi}$ as follows:

$$\begin{aligned} \widehat{x = x_k} &:= p_{x_k} & \widehat{x_k = x_l} &:= \mathbf{EF}(p_{x_k} \wedge p_{x_l}) \\ \widehat{x \in X_k} &:= p_{x_k} & \widehat{x_k \in X_l} &:= \mathbf{EF}(p_{x_k} \wedge p_{X_l}) \\ \widehat{\neg\varphi'} &:= \neg\widehat{\varphi'} & \widehat{\varphi_1 \vee \varphi_2} &:= \widehat{\varphi_1} \vee \widehat{\varphi_2} \\ \widehat{\exists x_k. \varphi'} &:= \exists p_{x_k}. \text{uniqu}(p_{x_k}) \wedge \widehat{\varphi'} & & \\ \widehat{\exists X_k. \varphi'} &:= \exists p_{X_k}. \widehat{\varphi'} & & \\ \widehat{S(x, x_k)} &:= \mathbf{Ex} p_{x_k} & \widehat{S(x_k, x)} &:= \perp \\ \widehat{S(x_k, x_l)} &:= \mathbf{EF}(p_{x_k} \wedge \mathbf{Ex} p_{x_l}) & & \end{aligned}$$

where $\text{uniqu}(p) := \mathbf{EF}p \wedge \forall q. (\mathbf{EF}(p \wedge q) \rightarrow \mathbf{AG}(p \rightarrow q))$ holds in a tree iff it has exactly one node labelled with p . To understand the $x = x_k$ case, we will interpret x as the root. To understand the $S(x_k, x)$ case, observe that x has no incoming edge since it is interpreted as the root.

The rules for eq are as follows:

$$\begin{aligned} \widehat{\text{eq}(x, x_k)} &:= p_{x_k} \\ \widehat{\text{eq}(x_k, x_l)} &:= \exists^\emptyset p. \mathbf{border}(p) \wedge \mathbf{AG}(p_{x_k} \rightarrow p \wedge p_{x_l} \rightarrow p) \end{aligned}$$

To understand the first case, observe that since x is interpreted as the root, x_k is on the same level as x if and only if it is also assigned the root. To understand the second case, recall from Example 2 that the QCTL_{ii}^* formula $\exists^\emptyset p. \mathbf{border}(p)$ places one unique horizontal line of p 's in the tree, and thus requiring that x_k and x_l be both on this line ensures that they are on the same level.

To finish, let \mathcal{S} be a CKS with two states s_0 and s_1 (local states are irrelevant here), whose transition relation is the complete relation, and with empty labelling function. Clearly, $t_S(s_0)$ is the full binary tree. It is easy to prove the following by induction:

For every $\varphi(x, x_1, \dots, x_i, X_1, \dots, X_j) \in \text{MSO}_{\text{eq}}$, $t_S(s) \models \varphi(s, u_1, \dots, u_i, U_1, \dots, U_j)$ if and only if $t_S(s_0)', s \models \widehat{\varphi}$, where $t_S(s_0)'$ is obtained from $t_S(s_0)$ by changing the labelling for variables p_{x_k} and p_{X_k} as follows: $p_{x_k} \in \ell'(u)$ if $u = u_k$ and $p_{X_k} \in \ell'(u)$ if $u \in U_k$.

In particular, it follows that

$$t_S(s_0), s_0 \models \varphi(x) \text{ iff } t_S(s_0), s_0 \models \widehat{\varphi}.$$

This completes the proof.

1

D. Proof of Lemma 1

Before presenting the proof of Lemma 1 we recall the definition of acceptance by ATA via games between Eve and Adam. Let $\mathcal{A} = (Q, \delta, q_i, C)$ be an ATA over (AP, X) -trees, let $t = (\tau, \ell)$ be such a tree and let $u_t \in \tau$. We define the parity game $\mathcal{G}(\mathcal{A}, t, u_t) = (V, E, v_t, C')$: the set of positions is $V = \tau \times Q \times \mathbb{B}^+(X \times Q)$, the initial position is $v_t = (u_t, q_t, \delta(q_t, u_t))$, and a position (u, q, α) belongs to Eve if α is of the form $\alpha_1 \vee \alpha_2$ or $[x, q']$; otherwise it belongs to Adam. Moves in $\mathcal{G}(\mathcal{A}, t, u_t)$ are defined by the following rules:

$$(u, q, \alpha_1 \dagger \alpha_2) \rightarrow (u, q, \alpha_i) \quad \text{where } \dagger \in \{\vee, \wedge\} \text{ and } i \in \{1, 2\}$$

$$(u, q, [x, q']) \rightarrow (u \cdot x, q', \delta(q', \ell(u \cdot x)))$$

Positions of the form (u, q, \top) and (u, q, \perp) are deadlocks, winning for Eve and Adam respectively. The colouring is inherited from the one of the automaton: $C'(u, q, \alpha) = C(q)$.

A tree t is *accepted* in node u by \mathcal{A} if Eve has a winning strategy in $\mathcal{G}(\mathcal{A}, t, u)$.

We now recall the lemma and the automata construction, and we prove it to be correct.

Lemma 7 (Translation). *Let (Φ, \mathcal{S}) be an instance of the model-checking problem for $\text{QCTL}_{i,\subseteq}^*$. For every subformula φ of Φ and state s of \mathcal{S} , one can build an ATA \mathcal{A}_s^φ on $(\text{AP}_\exists(\Phi), L_\varphi)$ -trees such that for every $(\text{AP}_\exists(\Phi), L_\varphi)$ -tree t rooted in $s \downarrow_{I_\varphi}$,*

$$t \in \mathcal{L}(\mathcal{A}_s^\varphi) \text{ iff } t \upharpoonright_y^{[n]} \mathrel{\mathbb{M}} t_{\mathcal{S}}(s) \models \varphi, \quad \text{where } y = s \downarrow_{[n] \setminus I_\varphi}.$$

For an L_I -tree t , from now on $t \uparrow^{[n]} \mathbin{\text{\texttt{M}}\kern-1.5ex\text{\texttt{M}}} t_S(s)$ stands for $t \uparrow_y^{[n]} \mathbin{\text{\texttt{M}}\kern-1.5ex\text{\texttt{M}}} t_S(s)$, where $y = s \downarrow_{[n] \setminus I}$: while lifting tree t to $[n]$, the missing local states in the root of t are filled with those from s .

Proof sketch of Lemma 1. Let (Φ, \mathcal{S}) be an instance of the model-checking problem for $\text{QCTL}_{i,\subseteq}^*$. Let $\Phi \in \text{QCTL}_{i,\subseteq}^*$, and let $\text{AP}_\exists = \text{AP}_\exists(\Phi)$ and $\text{AP}_f = \text{AP}_f(\Phi)$, and recall that \mathcal{S} is labelled over AP_f . For each state $s \in S$ and each subformula φ of Φ (note that all subformulas of Φ are also hierarchical), we define by induction on φ the ATA \mathcal{A}_s^φ on $(\text{AP}_\exists, L_\varphi)$ -trees.

$$\varphi = p :$$

We let \mathcal{A}_s^p be the ATA over $L_{[n]}$ -trees with one unique state q_s , with transition function defined as follows:

$$\delta(q_\ell, a) = \begin{cases} \top & \text{if } \begin{array}{l} p \in \text{AP}_f \text{ and } p \in \ell_S(s) \\ \quad \text{or} \\ p \in \text{AP}_\exists \text{ and } p \in a \end{array} \\ \perp & \text{if } \begin{array}{l} p \in \text{AP}_f \text{ and } p \notin \ell_S(s) \\ \quad \text{or} \\ p \in \text{AP}_\exists \text{ and } p \notin a \end{array} \end{cases}$$

$$\varphi = \neg\varphi' :$$

We obtain \mathcal{A}_s^φ by dualising $\mathcal{A}_s^{\varphi'}$, a classic operation.

$$\varphi = \varphi_1 \vee \varphi_2 :$$

Because $I_\varphi = I_{\varphi_1} \cap I_{\varphi_2}$, and each $\mathcal{A}_s^{\varphi_i}$ works on L_{φ_i} -trees, we first narrow them so that they work on L_φ -trees: for $i \in \{1, 2\}$, we let $\mathcal{A}_i := \mathcal{A}_s^{\varphi_i} \downarrow_{I_\varphi}$. We then build \mathcal{A}_s^φ by taking the disjoint union of \mathcal{A}_1 and \mathcal{A}_2 and adding a new initial state that nondeterministically chooses which of \mathcal{A}_1 or \mathcal{A}_2 to execute on the input tree, so that $\mathcal{L}(\mathcal{A}_s^\varphi) = \mathcal{L}(\mathcal{A}_1) \cup \mathcal{L}(\mathcal{A}_2)$.

$$\varphi = \mathbf{E}\psi :$$

The aim is to build an automaton \mathcal{A}_s^φ that works on L_φ -trees and that on input t , checks for the existence of a path in $t \uparrow^{[n]} \wedge t_{\mathcal{S}}(s)$ that satisfies ψ . Observe that a path in $t \uparrow^{[n]} \wedge t_{\mathcal{S}}(s)$ is a path in $t_{\mathcal{S}}(s)$, augmented with the labelling for atomic propositions in AP_{\exists} carried by t .

To do so, \mathcal{A}_s^φ guesses a path λ in (\mathcal{S}, s) . It remembers the current state in \mathcal{S} , which provides the labelling for atomic propositions in AP_f , and while it guesses λ it follows its L_φ -narrowing in its input tree t (which always exists since input to tree automata are complete trees), reading the labels to evaluate propositions in AP_\exists .

Let $\max(\psi) = \{\varphi_1, \dots, \varphi_n\}$ be the set of maximal state sub-formulas of ψ . In a first step we see these maximal state sub-formulas as atomic propositions. The formula ψ can thus be seen as an LTL formula, and we can build a nondeterministic parity word automaton $\mathcal{W}^\psi = (Q^\psi, \Delta^\psi, q_i^\psi, C^\psi)$ over alphabet $2^{\max(\psi)}$ that accepts exactly the models of ψ [35].⁵ We define the ATA \mathcal{A} that, given as input a $(\max(\psi), L_\varphi)$ -tree t , nondeterministically guesses a path λ in $t \uparrow^{[n]} \mathbb{M} t_S(s)$ and simulates \mathcal{W}^ψ on it, assuming that the labels it reads while following $\lambda \downarrow_{I_\varphi}$ in its input correctly represent the truth value of formulas in $\max(\psi)$ along λ . Recall that $S = (S, R, s_t, \ell_S)$; we define $\mathcal{A} := (Q, \delta, q_t, C)$, where

- $Q = Q^\psi \times S$,
 - $q_t = (q_t^\psi, s)$,
 - $C(q^\psi, s') = C^\psi(q^\psi)$, and
 - for each $(q^\psi, s') \in Q$ and $a \in 2^{\max(\psi)}$,

$$\delta((q^\psi, s'), a) = \bigvee_{q' \in \Delta^\psi(q^\psi, a)} \bigvee_{s'' \in R(s')} [s'' \downarrow_{I_\varphi}, (q', s'')].$$

The intuition is that \mathcal{A} reads the current label, chooses nondeterministically which transition to take in \mathcal{W}^ψ , chooses a next state in \mathcal{S} and proceeds in the corresponding direction in X_φ . Thus, \mathcal{A} accepts a $\max(\varphi)$ -labelled L_φ -tree t iff there is a path in t that is the L_φ -narrowing of some path in $t_{\mathcal{S}}(s)$, and that satisfies ψ , where maximal state formulas are considered as atomic propositions.

Now from \mathcal{A} we build the automaton \mathcal{A}_s^φ over L_φ -trees labelled with “real” atomic propositions in AP_\exists . In each node it visits, \mathcal{A}_s^φ guesses what should be its labelling over $\max(\psi)$, it simulates \mathcal{A} accordingly, and checks that the guess it made is correct. If the path being guessed in $t_{\mathcal{S}}(s)$ is currently in node u ending with state s' , and \mathcal{A}_s^φ guesses that φ_i holds in

⁵Note that, as usual for nondeterministic word automata, we take the transition function of type $\Delta^\psi : Q^\psi \times 2^{\max(\psi)} \rightarrow 2^{Q^\psi}$.

u , it checks this guess by starting a simulation of automaton $\mathcal{A}_{s'}^{\varphi_i}$ from node $v = u \downarrow_{I_\varphi}$ in its input t .

For each $s' \in \mathcal{S}$ and each $\varphi_i \in \max(\psi)$ we first build $\mathcal{A}_{s'}^{\varphi_i}$, and we let $\mathcal{A}_{s'}^i := \mathcal{A}_{s'}^{\varphi_i} = (Q_{s'}^i, \delta_{s'}^i, q_{s'}^i, C_{s'}^i)$. We also let $\mathcal{A}_{s'}^i = (Q_{s'}^i, \delta_{s'}^i, q_{s'}^i, C_{s'}^i)$ be its dualisation, and we assume w.l.o.g. that all the state sets are pairwise disjoint. Observe that because each φ_i is a maximal state sub-formula, we have $I_{\varphi_i} = I_\varphi$, so that we do not need to narrow down automata $\mathcal{A}_{s'}^i$ and $\mathcal{A}_{s'}^i$. We define the ATA

$$\mathcal{A}_s^\varphi = (Q \cup \bigcup_{i,s'} Q_{s'}^i \cup \overline{Q_{s'}^i}, \delta', q_0, C'),$$

where the colours of states are left as they were in their original automaton, and δ is defined as follows. For states in $Q_{s'}^i$ (resp. $\overline{Q_{s'}^i}$), δ agrees with $\delta_{s'}^i$ (resp. $\overline{\delta_{s'}^i}$), and for $(q^\psi, s') \in Q$ and $a \in 2^{\text{AP}_\exists}$ we let

$$\begin{aligned} \delta'((q^\psi, s'), a) := \bigvee_{a' \in 2^{\max(\psi)}} & \left(\delta((q^\psi, s'), a') \wedge \right. \\ & \bigwedge_{\varphi_i \in a'} \delta_{s'}^i(q_{s'}^i, a) \wedge \\ & \left. \bigwedge_{\varphi_i \notin a'} \overline{\delta_{s'}^i}(q_{s'}^i, a) \right). \end{aligned}$$

$\varphi = \exists^0 p \cdot \varphi'$:

We build automaton $\mathcal{A}_s^{\varphi'}$ that works on $L_{\varphi'}$ -trees; because φ is hierarchical, we have that $\mathbf{o} \subseteq I_{\varphi'}$ and we can narrow down $\mathcal{A}_s^{\varphi'}$ to work on $L_{\mathbf{o}}$ -trees and obtain $\mathcal{A}_1 := \mathcal{A}_s^{\varphi'} \downarrow_{\mathbf{o}}$. By Theorem 6 we can nondeterminise it to get \mathcal{A}_2 , which by Theorem 5 we can project with respect to p , finally obtaining $\mathcal{A}_s^\varphi := \mathcal{A}_2 \Downarrow_p$.

Correctness. We now prove by induction on φ that the construction is correct. In each case, we let $t = (\tau, \ell)$ be a complete $(\text{AP}_\exists, L_\varphi)$ -tree rooted in $s \downarrow_{L_\varphi}$.

$\varphi = p$:

First, note that $I_p = [n]$, so that t is rooted in $s \downarrow_{L_p} = s$. Let us consider first the case where $p \in \text{AP}_f$. By definition of \mathcal{A}_s^p , we have that $t \in \mathcal{L}(\mathcal{A}_s^p)$ iff $p \in \ell_s(s)$. On the other hand, by definition of the merge operation, of the unfolding, and because AP_\exists and AP_f are disjoint, we have $t \uparrow^{[n]} \wedge t_s(s) \models p$ iff $p \in \ell_s(s)$, and we are done. Now if $p \in \text{AP}_\exists$: by definition of \mathcal{A}_s^p , we have $t \in \mathcal{L}(\mathcal{A}_s^p)$ iff $p \in \ell(s)$; also, by definition of the merge, of the unfolding, and because AP_\exists and AP_f are disjoint, we have that $t \uparrow^{[n]} \wedge t_s(s) \models p$ iff $p \in \ell(s)$, which concludes.

$\varphi = \neg\varphi'$:

This case is trivial.

$\varphi_1 \vee \varphi_2$:

We have $\mathcal{A}_i = \mathcal{A}_{s'}^{\varphi_i} \downarrow_{L_\varphi}$, so by Theorem 7, for all $y \in L_{I_{\varphi_i} \setminus I_\varphi}$, we have $t \in \mathcal{L}(\mathcal{A}_i)$ iff $t \uparrow_y^{L_{\varphi_i}} \in \mathcal{L}(\mathcal{A}_{s'}^{\varphi_i})$, which by

induction hypothesis holds iff $(t \uparrow_y^{L_{\varphi_i}}) \uparrow^{[n]} \wedge t_s(s) \models \varphi_i$. Choosing $y = s \downarrow_{I_{\varphi_i} \setminus I_\varphi}$, we get that $t \in \mathcal{L}(\mathcal{A}_i)$ iff $t \uparrow^{[n]} \wedge t_s(s) \models \varphi_i$. We conclude by reminding that $\mathcal{L}(\mathcal{A}_s^\varphi) = \mathcal{L}(\mathcal{A}_1) \cup \mathcal{L}(\mathcal{A}_2)$.

$\varphi = \mathbf{E}\psi$:

Suppose that $t' = t \uparrow^{[n]} \wedge t_s(s) \models \mathbf{E}\psi$. There exists a path λ starting at the root s of t' such that $t', \lambda \models \psi$. Again, let $\max(\psi)$ be the set of maximal state subformulas of ψ , and let w be the infinite word over $2^{\max(\psi)}$ that agrees with λ on the state formulas in $\max(\psi)$. By definition, \mathcal{W}^ψ has an accepting execution on w . Now in the acceptance game of \mathcal{A}_s^φ on t , Eve can guess the path λ , following $\lambda \downarrow_{X_\varphi}$ in its input t , and she can also guess the corresponding word w on $2^{\max(\psi)}$ and an accepting execution of \mathcal{W}^ψ on w . Let $u' \in t'$ be a node of λ , s' its last direction and let $u = u' \downarrow_{L_\varphi} \in t$. Assume that in node u of the input tree, in a state $(q^\psi, s') \in Q$, Adam challenges Eve on some $\varphi_i \in \max(\psi)$ that she assumes to be true in u' , i.e., Adam chooses the conjunct $\delta_{s'}^i(q_{s'}^i, a)$, where a is the label of u . Note that in the evaluation game this means that Adam moves to position $(u, (q^\psi, s'), \delta_{s'}^i(q_{s'}^i, a))$. We want to show that Eve wins from this position.

Let t_u (resp. $t'_{u'}$) be the subtree of t (resp. t') starting in u (resp. u')⁶. It is enough to show that t_u is accepted by $\mathcal{A}_{s'}^i = \mathcal{A}_{s'}^{\varphi_i}$. Observe that t_u is rooted in the last direction of $u = u' \downarrow_{L_\varphi}$, and since the last direction of u' is s' we have that t_u is rooted in $s' \downarrow_{L_\varphi}$. Since $I_\varphi = I_{\varphi_i}$ we have that t_u is rooted in $s' \downarrow_{L_{\varphi_i}}$. We can thus apply the induction hypothesis on t_u with φ_i , and we get that

$$t_u \in \mathcal{L}(\mathcal{A}_{s'}^{\varphi_i}) \text{ iff } t_u \uparrow^{[n]} \wedge t_s(s') \models \varphi_i. \quad (1)$$

Now, because u' ends in s' we also have that

$$t'_{u'} = t_u \uparrow^{[n]} \wedge t_s(s'). \quad (2)$$

Putting (1) and (2) together, we obtain that

$$t_u \in \mathcal{L}(\mathcal{A}_{s'}^i) \text{ iff } t'_{u'} \models \varphi_i. \quad (3)$$

Because we have assumed that Eve guesses w correctly, we also have that $t', u' \models \varphi_i$, i.e., $t'_{u'} \models \varphi_i$. This, together with (3), gives us that t_u is accepted by $\mathcal{A}_{s'}^i$.

Eve thus has a winning strategy from the initial position of the acceptance game of $\mathcal{A}_{s'}^i$ on t_u . This initial position is $(u, q_{s'}^i, \delta_{s'}^i(q_{s'}^i, a))$. Since $(u, q_{s'}^i, \delta_{s'}^i(q_{s'}^i, a))$ and $(u, q, \delta_{s'}^i(q_{s'}^i, a))$ contain the same node u and transition formula $\delta_{s'}^i(q_{s'}^i, a)$, a winning strategy in one of these positions⁷ is also a winning strategy in the other, and therefore Eve wins Adam's challenge. With a similar argument, we get that also when Adam challenges Eve on some φ_i assumed not to be true in node v , Eve wins the challenge. Finally, Eve wins the acceptance game of \mathcal{A}_s^φ on t , and thus $t \in \mathcal{L}(\mathcal{A}_s^\varphi)$.

For the other direction, assume that $t \in \mathcal{L}(\mathcal{A}_s^\varphi)$, i.e., Eve wins the evaluation game of \mathcal{A}_s^φ on t . Again, let $t' = t \uparrow^{[n]}$

⁶If $u = w \cdot x$, a subtree t_u of $t = (\tau, \ell)$ is defined as $t_u := (\tau_u, \ell_u)$ with $\tau_u = \{x \cdot w' \mid w \cdot x \cdot w' \in \tau\}$, and $\ell_u(x \cdot w') = \ell(w \cdot x \cdot w')$: we remove from each node all directions before last(u).

⁷Recall that positional strategies are sufficient in parity games [38].

$\wedge t_S(s)$. A winning strategy for Eve describes a path λ in $t_S(s)$, which is also a path in t' . This winning strategy also defines an infinite word w over $2^{\max(\psi)}$ such that w agrees with λ on the formulas in $\max(\psi)$, and it also describes an accepting run of \mathcal{W}^ψ on w . Hence $t', \lambda \models \psi$, and $t' \models \varphi$.

$$\varphi = \exists^0 p. \varphi' :$$

First, observe that because φ is hierarchical, we have that $I_\varphi = \mathbf{o}$. Next, by Theorem 5 we have that

$$t \in \mathcal{L}(\mathcal{A}_s^\varphi) \text{ iff there exists } t_p \equiv_p t \text{ s.t. } t_p \in \mathcal{L}(\mathcal{A}_2). \quad (4)$$

By Theorem 6, $\mathcal{L}(\mathcal{A}_2) = \mathcal{L}(\mathcal{A}_1)$, and since $\mathcal{A}_1 = \mathcal{A}_s^{\varphi'} \downarrow_{\mathbf{o}} = \mathcal{A}_s^{\varphi'} \downarrow_{L_\varphi}$ we get by Theorem 7 that

$$t_p \in \mathcal{L}(\mathcal{A}_2) \text{ iff } t_p \uparrow_y^{L_{\varphi'}} \in \mathcal{L}(\mathcal{A}_s^{\varphi'}), \text{ where } y = s \downarrow_{(I_{\varphi'} \setminus I_\varphi)}. \quad (5)$$

Now t_p and t have the same root, $s \downarrow_{L_\varphi}$. The root of $t_p \uparrow_y^{L_{\varphi'}}$ is thus $(s \downarrow_{L_\varphi}, y) = s \downarrow_{L_{\varphi'}}$, and we can apply the induction hypothesis on $t_p \uparrow_y^{L_{\varphi'}}$ with φ' :

$$t_p \uparrow_y^{L_{\varphi'}} \in \mathcal{L}(\mathcal{A}_s^{\varphi'}) \text{ iff } (t_p \uparrow_y^{L_{\varphi'}}) \uparrow^{[n]} \wedge t_S(s) \models \varphi'. \quad (6)$$

Now, combining points (4), (5) and (6), together with the fact that $(t_p \uparrow_y^{L_{\varphi'}}) \uparrow^{[n]} \wedge t_S(s) = t_p \uparrow^{[n]} \wedge t_S(s)$ gives us that

$$t \in \mathcal{L}(\mathcal{A}_s^\varphi) \text{ iff } \exists t_p \equiv_p t \text{ s.t. } t_p \uparrow^{[n]} \wedge t_S(s) \models \varphi'. \quad (7)$$

We now prove equation below which, together with point (7), concludes the proof:

$$\begin{aligned} \exists t_p \equiv_p t \text{ s.t. } t_p \uparrow^{[n]} \wedge t_S(s) &\models \varphi' \\ \text{iff} \\ t \uparrow^{[n]} \wedge t_S(s) &\models \exists^0 p. \varphi' \end{aligned} \quad (8)$$

For the first direction, assume that there exists $t_p \equiv_p t$ such that $t_p \uparrow^{[n]} \wedge t_S(s) \models \varphi'$. First, by definition of the merge, because $p \in AP_\exists$ and AP_\exists and AP_f are disjoint, the p -labelling of $t_p \uparrow^{[n]} \wedge t_S(s)$ is determined by the p -labelling of $t_p \uparrow^{[n]}$, which by definition of the widening is \mathbf{o} -uniform. In addition it is clear that $t_p \uparrow^{[n]} \wedge t_S(s) \equiv_p t \uparrow^{[n]} \wedge t_S(s)$, which concludes this direction.

For the other direction, assume that $t \uparrow^{[n]} \wedge t_S(s) \models \exists^0 p. \varphi'$: there exists $t'_p \equiv_p t \uparrow^{[n]} \wedge t_S(s)$ such that t'_p is \mathbf{o} -uniform in p and $t'_p \models \varphi'$. Let us write $t'_p = (\tau', \ell'_p)$ and $t = (\tau, \ell)$. We define $t_p := (\tau, \ell_p)$ where for each $u \in \tau$, if there exists $u' \in \tau'$ such that $u' \downarrow_{\mathbf{o}} = u$, we let

$$\ell_p(u) = \begin{cases} \ell(u) \cup \{p\} & \text{if } p \in \ell'_p(u') \\ \ell(u) \setminus \{p\} & \text{otherwise.} \end{cases}$$

This is well defined because t'_p is \mathbf{o} -uniform in p : if two nodes u', v' project on u , we have $u' \approx_{\mathbf{o}} v'$ and thus they agree on p . In case there is no $u' \in \tau'$ such that $u' \downarrow_{L_\varphi} = u$, we can let $\ell_p(u) = \ell(u)$ as this node disappears in $t_p \uparrow^{[n]} \wedge t_S(s)$. Clearly, $t_p \equiv_p t$. Now we write $t''_p = t_p \uparrow^{[n]} \wedge t_S(s)$ and we prove that $t''_p = t'_p$ hence $t''_p \models \varphi'$, which concludes. It is clear that t''_p and t'_p have the same domain. Also, because $t'_p \equiv_p t \uparrow^{[n]} \wedge t_S(s)$ and $t''_p = t_p \uparrow^{[n]} \wedge t_S(s)$, by definition of

the merge both agree with $t_S(s)$ for all atomic propositions in AP_f . Because $t_p \equiv_p t$, and again by definition of the merge, t''_p and t'_p also agree on all atomic propositions in $AP_\exists \setminus \{p\}$. Finally, by definition of t_p and because t'_p is \mathbf{o} -uniform in p , we get that t''_p and t'_p also agree on p , and therefore $t''_p = t'_p$. \square