Computational Aspects of Infinite-State Verification

Cumulative Habilitation Thesis

by Stefan Göller

Defended on February 5th 2014

Contents

1 Introduction

Infinite-state systems arise in many areas of computer science. Typical aspects one wishes to model by them include the recursive behavior of programs, abstract data types like queues, the communication between unbounded buffers or the real-time behavior of systems, numeric data types like the integers or the reals, and many more.

Technically speaking we employ some finite encoding for representing these infinitely many states and infinite relations between them. For instance, Petri nets allow to model certain concurrent aspects of systems – their finite description consists of the set of places and of the transitions that can be fired between them.

The design and analysis of infinite-state systems has attracted a lot of research in the last twenty years and indeed various models of infinite-state systems have been introduced and studied in the literature such as the above-mentioned Petri nets, (higher-order) pushdown systems, well-structured transition systems, automatic structures, ground tree rewrite systems and counter systems, just to mention a small fraction of them.

Two important aspects of the models of infinite-state systems are the following. On the one hand, one wants the model to be as expressive as possible in order to model as many systems as accurately as possible. On the other hand, one wishes to retain decidability and a low complexity with respect to algorithmic verification problems such as reachability. One can easily see that there is a trade-off between the two. For instance, if our infinite-state model allows to encode the configuration graph of any Turing machine, we can be sure that reachability is undecidable.

In the context of hardware and software systems, *formal verification* is the act of proving or disproving the correctness of intended behavior of a system with respect to a certain formal specification or property using formal methods.

Model checking is a fully-automatic formal verification method which has been proven successful in validating and verifying safety-critical systems. It asks to decide whether a given system satisfies a given property, where the property is typically specified in some suitable logic. If we would like to know whether the system satisfies a particular property, we construct an abstract model $\mathcal{T}$ for the system that comprises the behavior of the system and we express the property via a formula φ in some logical language. Hence, we solve the initial problem by checking whether $\mathcal{T}$ satisfies φ, which in turn can be checked by using efficient model checking algorithms. Concerning the choice of the suitable logic similar trade-off remarks apply: on the one hand, the logic should be sufficiently expressive for being able to capture the relevant properties as precisely as possible – on the other hand, model checking the logic should at least be decidable and should have a low complexity, if possible.

Vardi proposed three different ways of measuring the complexity of the model checking problem [191] that we summarize briefly. Often, the presentation of the system is much larger than the size of the formula. This motivates to study the *data complexity* of model checking, which views the formula as fixed and measures the computational complexity only in terms of the size

of the input system; thus one has a decision problem for each fixed formula. However, it can very well be the case that one is aware of the system itself and has varying formulas to be verified. In this spirit, the *expression complexity* assumes the system to be fixed and asks to decide whether it models the input formula. When both the system and the formula are accounted for as non-constant and are hence part of the input, one obtains the most general variant, the *combined complexity* of the model checking problem.

From a complexity-theoretic viewpoint it seems fair to say that model checking *finite* systems is by now very well understood. A prominent exception is surely the complexity of model checking the modal μ-calculus: the problem is known to lie in $\mathsf{UP} \cap \mathsf{coUP}$ [111] and hard for deterministic polynomial time.

Over infinite-state systems however, already much simpler questions such as reachability can indeed be much more involved. A very prominent example is the reachability problem for Petri nets: to date the problem is known to be decidable [117, 136], but the best-known lower bound is Lipton's EXPSPACE-hardness proof from the seventies [37]. The problem is not even known to be primitive-recursive.

On the other hand, powerful tools have been developed for obtaining decidability results. Rabin's tree theorem is surely one of the central decidability results in this context: it states that the monadic second-order (MSO) theory of the infinite binary tree is decidable. This result can immediately be applied to model checking pushdown systems (which are the transition systems induced by pushdown automata), which provide a very natural way of modeling the call and return behavior of recursive programs. Muller and Schnupp were the first to see that Rabin's tree theorem implies that model checking MSO on pushdown systems is decidable [146]: every pushdown system is interpretable in the complete binary tree via MSO formulas. It is worth mentioning that monadic second-order logic (the extension of first-order logic by allowing to quantify over sets of elements of the domain instead of just over elements themselves) is indeed a very powerful logic. Undeniably, this expressiveness comes at a price: the monadic second-order theory of the infinite binary tree is nonelementary and thus the same nonelementary lower bound applies to MSO model checking pushdown systems.

Temporal logics have caught more and more attention as specification formalisms in the last forty years not only in the context of model checking. The reason for this is that temporal logics are still expressive enough for modeling the relevant behaviors of systems and moreover, in contrast to powerful logics like the above-mentioned monadic second-order logic, the complexity of model checking decreases dramatically. The term "temporal logic" is used to describe logical means for representing, and reasoning about, propositions qualified in terms of time. Two classical such properties one wishes to express are statements like "every request is eventually met by a response" or "one will never reach a deadlock". Three classical examples of temporal logics include Linear Temporal Logic (LTL) [153] in which formulas make statements about the future of paths, Computation Tree Logic (CTL) [47] in which time is accounted for in a tree-like fashion, and the modal μ-calculus ([4]) that extends classical modal logic by adding least and greatest fixed point operators.

It has been shown by Walukiewicz that model checking the modal μ-calculus on pushdown systems is EXP-complete [197], whereas Kupferman and Vardi have followed an automata-theoretic approach for obtaining such an exponential time decision procedure [120]. The latter was inspired by Vardi's EXP-completeness result on emptiness of two-way alternating parity

tree automata [190]. Bouajjani, Esparza and Maler proved that LTL model checking over pushdown systems is EXP-complete [18]. Moreover, in [18] it has been shown that from a regular set of configurations both the set of reachable configurations and the set of configurations that can reach this set are effectively regular again and furthermore computable in polynomial time. Even better, the data complexity of model checking LTL over pushdown systems is decidable in polynomial time, where it is still EXP-hard for the modal μ-calculus [196]. In fact, the latter EXP lower bound already holds for the logic CTL, whereas model checking CTL's fragment EF is only PSPACE-complete [196]. Model checking Propositional Dynamic Logics on pushdown systems and related models has been investigated in [81].

An important sublass of pushdown systems are the transition graphs of pushdown automata over a singleton set of control states, the so-called *basic process algebras* (BPA). Model checking various temporal logics on basic process algebras is often still (almost as) hard as for pushdown systems, however the data complexity dramatically decreases over them: it turns out to be decidable in polynomial time [139].

A further important subclass of pushdown systems are *one-counter systems*: here, the involved pushdown automata may contain an arbitrary finite set of control states, but there may only be one stack symbol (plus an additional bottom-of-stack symbol). In fact, one-counter systems can be viewed as being obtained from a certain finite system (that corresponds to the case when the counter is zero) that is connected to one of infinitely many successive copies of a further finite system (that corresponds to the positive counter values) that are connected with each other successively. Hence, the class of one-counter systems can be seen as one of the simplest means to model infinite-state systems. Serre proved that model checking the modal μ-calculus is PSPACE-complete on one-counter systems [168]. It can easily be seen that model checking LTL on one-counter systems is interreducible to model checking LTL on finite systems, and is in fact PSPACE-complete. EF model checking on one-counter systems has not been understood very well so far: it has been shown hard for the complexity class DP in [108].

In the last twenty years many generalizations of pushdown systems have been investigated. Pushdown systems themselves can analogously be seen as systems whose states are given by the set of all finite words (over some fixed finite alphabet) and whose transitions are induced by application of a finite set of word rewriting rules that are used in a prefix rewrite fashion. When these rewrite rules are generalized in such a way that not only words, but words inside some regular language (on both sides of each rule) are rewritten, one obains the class of *prefix-recognizable systems*. In fact, Caucal showed that prefix-recognizable systems have decidable MSO theories [40, 41]. Model checking the modal μ-calculus is still EXP-complete for them [33, 120], whereas already reachability becomes EXP-hard (when there is a suffix language involved in the rewrite rules) [71]. Caucal further generalized decidability of MSO model checking on the class of systems one obtains by alternately applying the operations unfolding and inverse rational mapping: the latter class is also known as the *Caucal hierarchy* [38]. The Caucal hierachy has been characterized by Carayol and Wöhrle [36] in terms of the transition graphs of higher-order pushdown automata (introduced by Maslov [134]). Usual pushdown automata manipulate usual stacks of atomic symbols, whereas the stacks of order-n pushdown automata consist of a sequence of order-(n-1) stacks for each $n > 1$. In fact, model checking various temporal logics on order-n pushdown systems quickly becomes nonelementary in n (even reachability), we refer to [89] for various results on model checking higher-order pushdown systems.

The before-mentioned classes provide good means for modeling the sequential behavior of systems. The class of Petri nets can be seen as the corresponding parallel analog of pushdown systems. Apart from reachability, decidability can moreover be shown for model checking LTL on Petri nets, but the problem is still at least as hard as the reachability problem [60]. In the same paper [60] Esparza proves that undecidability for model checking Petri nets already holds for the fragment EF of CTL. The complexity drops down to PSPACE-completeness for model checking EF over communication-free Petri nets [139] as shown by Mayr, whereas the complexity of reachability is much lower than for general Petri nets: it is NP-complete [61]. The latter class of communication-free Petri nets is also known as *basic parallel processes* (BPP).

In fact, Mayr found an elegant symbiosis of systems that behave sequentially or concurrently, or both: the process rewrite systems (PRS) hierarchy [140]. It combines systems whose states are essentially terms that can be built from basic atoms and applying the two operators sequential and parallel composition, respectively. The states of such a (general) PRS system consist essentially of the set of all such terms, but are interpreted in such a way that sequential composition is associative and parallel composition is both associative and commutative. The transitions can be seen to evolve from applying these term rewrite rules to subterms. Subclasses of PRS are given by putting syntactic restrictions on the left-hand side and right-hand side the rewrite rules to be either purely sequential, purely parallel, singletons or unrestricted.

Viewed in this way, pushdown systems (resp. Petri nets) are PRS in which both the left-hand side and the right-hand side of the rewrite rules are purely sequential (resp. purely parallel). Basic process algebras (resp. basic parallel processes) are the restrictions to those PRS, where the left-hand side is unary and the right-hand side is purely sequential (resp. purely parallel). Having mixed forms, where the right-hand side of rules is unrestricted but the left-hand side is possibly restricted, make up the the classes PA, PAD and PAN. Mayr showed that EF model checking on PAD is decidable [138]. For the class PA Lugiez and Schnoebelen proved decidability of model checking first-order logic with reachability [130]. However, undecidability of model checking EF holds for PAN since it is inherited from its undecidability over Petri nets. With the latter formalisms one can model systems that involve parallel programs with unbounded recursions and unbounded parallelism [5, 63].

A similar concept of defining infinite-state systems arose from the term rewriting community with the study of *ground tree rewrite systems* (GTRS) [49]. While pushdown systems can be seen as prefix word rewriting systems, the states of ground tree rewrite systems consist of finite ranked trees, where the transitions are induced by a finite set of ranked tree rewriting rules (that are applied to one subtree). The same way prefix-recognizable systems relate to pushdown systems, so do *regular ground tree rewrite systems* (RGTRS) relate to ground tree rewrite systems: instead of only allowing single trees in the rewrite rules, they allow regular tree languges to appear there. We refer to the work of Colcombet [48] for an algebraic treatment of them. The crucial difference to Mayr's PRS hierarchy is that the states of the underlying system that are defined by them are indeed the set of finite ranked trees and not equivalence classes on them (in PRS different terms can potentially represent the same state since one works modulo associativity/commutativity). For regular ground tree rewrite systems model checking first-order logic with reachability is decidable [56, 48]. Moreover, Löding showed that recurrent reachability and model checking EF (and several variants thereof) is decidable, whereas already model checking CTL's fragment EG (constrained reachability) becomes undecidable [128].

Apart from model checking, a second important verification task involves *equivalence checking,* which asks to determine whether two given systems behave equivalently with respect to some notion of equivalence, such as isomorphism for instance. So instead of having the specification given by some logical formalism, the desired behavior is rather given by some further system and one wishes to decide if this further system behaves equivalently to the system that is to be verified.

In particular, with the aid of an automated equivalence checker, it is possible for a system designer to replace complex systems by simpler system. When doing so, one might be able to drastically decrease the running time for verifying typical properties of interest and indeed make use of the specific properties of the simpler one. For instance, if one can be sure that a given infinite system behaves equivalently to a finite one, one could compute concrete response times instead of having to traverse an infinite-state space, which might be very inefficient.

Various notions of equivalences have been proposed in the literature [8], ranging from trace equivalence to isomorphism [69, 70]. Among these numerous notions of equivalence in verification, *bisimulation equivalence* is surely the central one, see e.g [177] for a survey. Elegant characterizations of well-known temporal logics have been proven in terms of bisimulation-invariant fragments of classical logics. A famous result due to van Benthem states that the bisimulation-invariant properties of first-order logic are precisely the properties that can be expressed in modal logic [187]. In the same spirit, the modal μ-calculus has been characterized as the bisimulation-invariant properties that can be expressed in monadic second-order logic due to Janin and Walukiwicz [97]. Finally, we mention a result by Moller and Rabinovich [145] who showed that the temporal logic CTL^* coincides with the bisimulation-invariant fragment of monadic path logic (which is the restriction of monadic second-order logic restricted to sets of elements of the domain that lie on a path).

Moreover, bisimulation equivalence has an vivid characterization in terms of a*game* played by two players "Attacker" and "Defender" who alternately move in a pebble bisimulation game on the pair of systems under consideration. One can prove that two systems are bisimulation equivalent (*bisimilar* for short) if, and only if, Defender has a winning strategy in this bisimulation game [181, 176]. In other words, the bisimulation game can be seen as a guarded variant of the classical Ehrenfeucht-Fraïsée game for first-order logic.

While between finite systems it is well-known that for bisimilarity checking efficient algorithms exist [150, 113] and that the problem is generally complete for deterministic polynomial time [7], only very little is known about the decidability and complexity status of equivalence checking on various classes of infinite-state systems. Concerning infinite-state systems, when comparing the knowledge and results that have been obtained on model checking with the ones for equivalence checking, it seems fair to claim that the understanding of equivalence checking can be assessed as premature in total. Indeed, on *any* of the above-mentioned classes of infinite-state systems, there has only been a single such class for which bisimilarity checking is known to be decidable and for which the precise complexity could be determined: Jančar proved PSPACE-completeness for bisimilarity checking of basic parallel processes [101]. Although this is subjective, it seems that a possible reason for the latter is that equivalence checking on infinite-state systems is a combinatiorally highly nontrivial problem. A summary of up-to-date records of results on equivalence checking of infinite-state systems in Mayr's PRS hierarchy [140] is

being maintained by Jiri Srba [174].

Despite the fact that in the field of infinite-state equivalence checking there are still many more techniques to be developed and more understanding to be gained, there are undoubtedly several highlights that should be mentioned (without claiming completeness). The most prominent result in this area is the decidability of equivalence of *deterministic pushdown automata (DPDA)*; this long-standing decidability question in formal languages was positively answered by Sénizergues [164] (see also [165]), for which Stirling [178] proved a primitive recursive upper bound. The problem still does not seem to be completely understood, which was one of the motivating factors for a recent simplified proof via first-order grammars, given in [103]. We note that the decidability status of language equivalence of deterministic higher-order pushdown systems remains an interesting open problem; some progress in this direction has been made by Stirling [179]. Regarding the lower bound, the DPDA language equivalence is only known P-hard (easily derivable from P-hardness of the emptiness problem), hence the known complexity gap is very large. To the best of the author's knowledge, we have the same phenomenon even for *real-time* DPDA [149], i.e. for DPDA in which ε-transitions are not present. A coNP upper bound was shown for finite-turn DPDA [166]. For simple grammars (real-time DPDA with a single control state), a polynomial algorithm deciding equivalence was shown in [94] (see [53] for a recent upper bound).

Sénizergues has lifted his decidability techniques to prove that bisimilarity of equational graphs (they lie between pushdown graphs and prefix-recognizable graphs) of finite out-degree is decidable [167]. EXP-hardness by Mayr and Kučera [121] was the best-known lower bound for this problem, yet Kiefer recently established EXP-hardness already for the class of basic process algebras [114], for which in turn at least a doubly exponential upper bound is known for a while [31]; we refer to [102] for a more rigorous and simpler proof.

Decidability of bisimilarity for one-counter systems is surely inherited from its decidability for pushdown systems, however Jančar independently established decidability in [100] whose algorithm has been analyzed to run in triply exponential space by Yen [200]. A PSPACE lower bound for bisimilarity of one-counter systems has been proven by Srba [175].

Concerning parallel models of computation Jančar proved that bisimilarity for Petri nets is undecidable [99]. For normed PA processes Jerrum and Hirshfeld proved that bisimilarity [93]is decidable in nondeterministc doubly exponential time, but decidability of the general case remains open.

Being used in most of the above-mentioned lower bounds proofs, a generic technique entitled "Defender's Forcing" has been developed by Jančar and Srba in [110], where it is demonstrated on the results like the undecidability (Π_1^0-completeness) of bisimilarity of pushdown systems with popping ε-steps or Σ_1^1-completeness on the class of prefix-recognizable systems. When reducing from a hard problem, the essential idea of "Defender's Forcing" tries to set up a bisimulation game that is designed in such a way that the pair of states/processes/configurations of the infinite system are almost always syntactically equivalent, for allowing to implement a gadget for Defender to make choices when necessary. Intuitively, due to the nature of the bisimulation game, Attacker generally has more freedom in his moves since he is the one who chooses the first system and the first transition in each round of the game. By forcing the pairs of states (which are pairs of words in pushdown systems for instance) the technique implements a particular gadget that gives Defender the possibility to make choices. Whether a conceptually different

technique can be developed – in particular for deterministic systems – is a challenging research question.

1.1 Contributions and organization of this thesis

This thesis summarizes several contributions of the author in the field of model checking and equivalence checking of infinite-state systems.

More concretely, we mainly study the model checking problem for the branching-time logics CTL, EF and Hennessy Milner logic HM and moreover equivalence checking (bisimulation equivalence, weak bisimulation equivalence, branching bisimulation equivalence and trace equivalence) on (a subset of) the following classes of systems: pushdown systems, one-counter systems as well as succinct and parametric one-counter systems, higher-order pushdown systems, ground tree rewrite systems, basic process algebras and PA/PAD-processes.

The submitted papers of this thesis are listed in Chapter 10. The contributions of the author in each chapter of this thesis are summarized in the Appendix.

Finally, we summarize the contents and results of this thesis:

- In Chapter 2 we provide some basic notation, introduce the relevant logics (CTL and its fragment EF and its fragment Hennessy Milner logic HM), introduce the different notions of equivalence we look at in this thesis, and define the central decision problems we study in this work.

- Chapter 3 surveys the classes of infinite-state systems that we concern ourselves with in this thesis. We give an overview of Mayr's Process Rewrite Systems (PRS) hierarchy and integrate one-counter systems, higher-order pushdown systems and ground tree rewrite systems into this hierarchy with respect to bisimilarity, weak bisimilarity and branching bisimilarity. While the former integration of one-counter systems and higher-order pushdowns is trivial, the integration of ground tree rewrite systems is less obvious. We discuss several relevant results in the literature on model checking and equivalence checking in all of these classes of systems.

  The main results in Section 3 appear only in Section 3.1 and consist of the following:

  – An integration of (regular) ground tree rewrite systems into Mayr's PRS hierarchy with respect to bisimilarity, weak bisimilarity and branching bisimilarity.

  These results will appear in Transactions on Computational Logic [78] and have been published as a conference paper in CONCUR 2011 [75] and are based on joint work with Anthony Widjaja Lin.

- In Chapter 4 is about equivalence checking of one-counter systems. The main results are the following:

  (1) It is shown that bisimulation equivalence of one-counter systems is complete for PSPACE (Theorem 4.2). This improves a previously best-known 3EXPSPACE upper bound for this problem and matches a PSPACE lower bound proven by Srba

[175]. Moreover, it witnesses one of very few classes of infinite-state systems, where bisimulation equivalence is decidable and moreover the precise computational complexity is known. Moreover, we show that deciding whether a one-counter system is bisimilar to a finite system is P-complete (Theorem 4.5), which improves a previously best-known upper bound of triply exponential time for this problem from [100, 200]. These results have been obtained in a joint work with Stanislav Böhm and Petr Jančar published in CONCUR 2010 [15].

(2) With a similar technique as in (1) it is shown that trace equivalence of deterministic real-time one-counter automata is NL-complete (Theorem 4.3). Moreover, we prove that deciding whether a deterministic real-time one-counter automaton accepts a regular language is NL-complete as well (Theorem 4.6). Both results improve a previously best-known $2^{O(\sqrt{n \log n})}$ time bounded algorithm from 1975 [186] for both problems. This result has been obtained in joint work with Stanislav Böhm published in MFCS 2011 [14].

Both results (1) and (2) have been merged into a journal paper that has been accepted for publication in Journal of Computer and System Sciences [16].

(3) We show that equivalence of deterministic one-counter automata is NL-complete (Theorem 4.27). The previously best-known upper bound for this problem is again the (already above-mentioned) algorithm of Valiant and Paterson running in time $2^{O(\sqrt{n \log n})}$ from 1975 [186], from which one can derive a PSPACE upper bound that has been the previously best-known complexity bound for this problem. This result has been obtained in a joint work with Stanislav Böhm and Petr Jančar published in STOC 2013 [17].

- In Chapter 5 we discuss the following results:

  (1) We show that there is already a fixed one-counter system for which CTL model checking PSPACE-hard (Theorem 5.4).

  (2) We "complement" Theorem 5.5 and show that model checking fixed one-counter systems with input CTL formulas of fixed leftward until depth is decidable in polynomial time (Theorem 5.5).

  (3) We develop a novel technique for proving lower bounds in model checking and reachability questions on transition systems induced by one-counter automata and timed automata. This technique was inspired by the question what the complexity of model checking one-counter systems with respect to fixed CTL formulas is. Inspired by two deep results from complexity theory, we develop a generic lower bound technique (Theorem 5.9) that allows us to derive the following hardness results:

    (3a) There exists a fixed CTL formula for which model checking one-counter systems is PSPACE-hard (Theorem 5.10).

    (3b) There exists a fixed CTL formula for which model checking succinct one-counter systems is EXPSPACE-hard (Theorem 5.12).

    (3c) Model checking CTL's fragment EF on one-counter systems is P^{NP}-hard (Theorem 5.11).

(3d) Deciding if a one-counter Markov decision process can reach a designated set of zero configurations with probability arbitrarily close to 1 is PSPACE-hard (Theorem 5.13).

(3e) Model checking 2-clock timed automata with constants presented in unary against fixed CTL formulas is PSPACE-hard (Theorem 5.15) and the reachability problem of 2-clock timed automata with very simple modulo tests and constants presented in unary is PSPACE-hard (Theorem 5.16).

The results have been obtained in a joint work with Markus Lohrey published in STACS 2012 [79] and will appear in SIAM Journal of Computing [80]. The only exception is result (3b) which has been obtained in a joint work with Christoph Haase, Joël Ouaknine and James Worrell published in ICALP 2010 [73].

- In Chapter 6 we discuss the computational complexity of model checking one-counter systems and succinct and parametric one-counter systems against the logics EF and Hennessy-Milner logic HM. Our results are the following:

  (1) Model checking EF on one-counter systems is in P^{NP} (Corollary 6.9). For this, we develop a suitable fragment of Presburger arithmetic that is tailored towards expressing the set of natural numbers that satisfy a given EF formula in a given control state of the one-counter system and provides a formalism for solving the global model checking problem.

  (2) Model checking EF on succinct one-counter systems is PSPACE-complete (Proposition 6.13 and Theorem 6.16).

  (3) Model checking EF on parametric one-counter systems is undecidable (Theorem 6.17).

  (4) Model checking HM on parametric one-counter systems is PSPACE-complete (Proposition 6.13 and Theorem 6.21).

  Result (1) has been obtained in a joint work with Anthony Widjaja To and Richard Mayr published in LICS 2009 [82] and the other results have been obtained in joint work with Christoph Haase, Joël Ouaknine and James Worrell published in FOSSACS 2012 [74].

- In Chapter 7 we concern ourselves with model checking EF (resp. HM) on the asynchronous product of basic process algebras (resp. of prefix-recognizable systems) and the sizes of Feferman-Vaught decompositions for EF and HM with respect to asynchronous product. The main results are the following:

  (1) Model checking EF on the asynchronous product of two basic process algebras is nonelementary (Theorem 7.2) and as a consequence model checking the asynchronous product of two prefix-recognizable systems is also nonelementary (Theorem 7.4). This solves questions raised by Löding and Mayr on the complexity of model checking EF on ground tree rewrite systems [128] and on PA/PAD processes [139], respectively.

  (2) The sizes of Feferman-Vaught type decompositions for EF and HM with respect to asynchronous product are inherently nonelementary (Theorem 7.5). The same

nonelementary lower bound holds when restricted to finite transition systems (Theorem 7.8).

These results have been obtained in a joint work with Anthony Widjaja Lin [77] published in STACS 2012.

- Chapter 8 provides lower bounds on the decidability and computational complexity of bisimilarity of pushdown systems and higher-order pushdown systems. We have the following results:
  (1) Bisimilarity of order-two pushdown systems is undecidable (Theorem 8.7). We also mention the undecidability of the lower order problem, i.e. deciding whether there exists a reachable configuration of an order-k pushdown system that is bisimilar to an order-k' system (Theorem 8.8). These results have been obtained in a joint work with Christopher Broadbent published in FSTTCS 2012 [27].
  (2) Bisimilarity of pushdown systems is nonelementary (Theorem 8.9). This result is a elaborate application of Defender's Forcing technique [110] and significanlty improves the previously best-known EXP lower bound of this problem due to Kučera and Mayr [121] which already dates back to 2002. This result has been obtained in a joint work with Michael Benedikt, Stefan Kiefer and Andrzej Murawski published in LICS 2013 [17].

- In Chapter 9 we study the computational complexity of model checking EF on (regular) ground tree rewrite systems. Our main results are the following:
  (1) Already model checking a given ground tree rewrite system against a given EF formula is nonelementary, already when the formula has two nestings of the EF operator (Theorem 9.1).
  (2) Model checking ground tree rewrite systems against EF formulas of EF nesting depth at most one is complete for the complexity class $\mathsf{P}^{\mathsf{NEXP}}$ (Corollary 9.9 and Theorem 9.9). The author is not aware of any previous natural problems that are complete for the complexity class $\mathsf{P}^{\mathsf{NEXP}}$. As an immediate corollary we obtain that checking bisimilarity between a ground tree rewrite system and a finite system is in coNEXP (Theorem 9.11), which provides a first elementary upper bound for this problem. The same results hold for PA processes.
  (3) Bisimilarity of regular ground tree rewrite systems and finite transition systems is nonelementary (Theorem 9.12). We apply a lower bound technique by Kučera and Mayr [121] in an elaborate way, where the main technical obstacle is the fact that regular ground tree rewrite systems are not closed under direct product with finite systems.

  These results have been obtained in a joint work with Anthony Widjaja Lin published in LICS 2011 [76].

2 Equivalence checking and model checking

By $\mathbb{N} = \{0, 1, \ldots\}$ we denote the set of non-negative integers and define $[i, j] \stackrel{\text{def}}{=} \{i, i+1, \ldots, j\}$ for each $i, j \in \mathbb{N}$. By $\mathbb{N}_+$ we denote the set $\mathbb{N} \setminus \{0\}$ of positive integers. For the rest of this document, let us fix a countable set of *atomic actions* Act. A *(labeled) transition system* is a tuple $\mathcal{T} = (S, \mathsf{A}, \{\stackrel{a}{\longrightarrow} | \ a \in \mathsf{A}\})$, where

- S is a set of *states*,
- $\mathsf{A} \subseteq \mathsf{Act}$ is a finite set of atomic actions, and
- $\stackrel{a}{\longrightarrow} \subseteq S \times S$ is a binary transition relation for each $a \in \mathsf{A}$.

We often write $s \stackrel{a}{\longrightarrow} s'$ to abbreviate $(s, s') \in \stackrel{a}{\longrightarrow}$ and just write $s \stackrel{a}{\longrightarrow}$ if there exists some state s' such that $s \stackrel{a}{\longrightarrow} s'$. The relations $\stackrel{a}{\longrightarrow}$ are extended to $\stackrel{w}{\longrightarrow}$ for words $w \in \mathsf{A}^*$ inductively: $s \stackrel{\varepsilon}{\longrightarrow} s$; if $s \stackrel{a}{\longrightarrow} s'$ and $s' \stackrel{u}{\longrightarrow} s''$ then $s \stackrel{au}{\longrightarrow} s''$. By $s \stackrel{w}{\longrightarrow}$ we denote that w is *enabled in* s, i.e. $s \stackrel{w}{\longrightarrow} s'$ for some $s' \in S$. For each $X \subseteq \mathsf{A}$, we define $\stackrel{X}{\longrightarrow} \stackrel{\text{def}}{=} \bigcup_{a \in X} \stackrel{a}{\longrightarrow}$. We write $\longrightarrow$ for $\stackrel{\mathsf{A}}{\longrightarrow}$ and by $\longrightarrow^*$ we denote the reflexive and transitive closure of $\longrightarrow$. Hence $s \longrightarrow^* s'$ if, and only if, $s \stackrel{w}{\longrightarrow} s'$ for some $w \in \mathsf{A}^*$, i.e. if, and only if, s' is *reachable from* s.

Notions of equivalence

The simplest and coarsest notion of equivalence that we would like to mention is *trace equivalence*. Two states s and s' are *trace equivalent* if $\{w \in \mathsf{A}^* \mid s \stackrel{w}{\longrightarrow}\} = \{w \in \mathsf{A}^* \mid s' \stackrel{w}{\longrightarrow}\}$.

Among the numerous notions of equivalence [188] in the realm of formal verification and concurrency theory, the central one is *bisimulation equivalence* (*bisimilarity* for short), which enjoys pleasant mathematical properties. It can be seen to take the king role: There are important characterizations of the bisimulation-invariant fragments of first-order logic, monadic second-order logic, and monadic path logic in terms of modal logic [187], the modal μ-calculus [97], and CTL^* respectively [145]. In particular, bisimilarity is a fundamental notion for process algebraic formalisms [143]. Many relevant properties of interests in verification (e.g. those expressible in standard modal/temporal logics like LTL,CTL, modal μ-calculus) cannot distinguish transition systems that are bisimilar.

A relation $R \subseteq S \times S$ is a *bisimulation* if R is symmetric and for each $(s, t) \in R$ and each $a \in \mathsf{A}$ the following holds:

- if $s \stackrel{a}{\longrightarrow} s'$ for some $s' \in S$, then $t \stackrel{a}{\longrightarrow} t'$ and $(s', t') \in R$ for some $t' \in S$.

We write $s \sim t$ if there exists some bisimulation R such that $(s,t) \in R$.

Bisimulations and weak bisimulations are historically the most important notions of bisimulations on transition systems in verification [144]. Weak bisimulations extend strong bisimulations by distinguishing observable and non-observable (i.e. τ) actions, and only require the observable behavior of two systems to agree. In this sense, weak bisimulation is a coarser notion than strong bisimulation.

Let us define $\stackrel{\tau}{\Longrightarrow} \stackrel{\text{def}}{=} \stackrel{\tau}{\longrightarrow}^*$ and $\stackrel{a}{\Longrightarrow} \stackrel{\text{def}}{=} \stackrel{\tau}{\longrightarrow}^* \circ \stackrel{a}{\longrightarrow} \circ \stackrel{\tau}{\longrightarrow}^*$ for each $a \in \mathsf{A} \setminus \{\tau\}$. A *weak bisimulation* with respect to some *internal symbol* $\tau \in \mathsf{A}$ (we also sometimes denote this symbol by ε in this thesis) is a symmetric relation $R \subseteq S \times S$ such that for each $(s,t) \in R$ and each $a \in \mathsf{A}$ the following holds:

- if $s \stackrel{a}{\longrightarrow} s'$ for some $s' \in S$, then $t \stackrel{a}{\Longrightarrow} t'$ and $(s',t') \in R$ for some $t' \in S$.

We write $s \approx t$ if there exists some weak bisimulation R such that $(s,t) \in R$.

Strong (resp. weak) bisimilarity can also be described by simple pebble games between two players: *Attacker* and *Defender*. Attacker's goal is to prove that two given states are *not* strongly (resp. *not* weakly) bisimilar, while Defender tries to prove otherwise. We will refer to Attacker as *him* and to Defender as *her*. In every round of the game, there is a pebble placed on a unique state in each transition system. Attacker then chooses one transition system and moves the pebble from the pebbled state to one of its successors by an action $\stackrel{a}{\longrightarrow}$, where $a \in \mathsf{A}$. Defender must imitate this by moving the pebbled state from the other system to one of its successors by the same action $\stackrel{a}{\longrightarrow}$ (resp. $\stackrel{a}{\Longrightarrow}$). If one player cannot move, then the other player wins. Defender wins every infinite game. Two states s and t are strongly/weakly bisimilar (resp. not strongly/weakly-bisimilar) if, and only if, Defender (resp. Attacker) has a winning strategy on the game with initial pebble configuration (s,t).

Branching bisimulation [189] is a notion of semantic equivalence that is strictly coarser than strong bisimulation but is strictly finer than weak bisimulation. It refines weak bisimulation equivalence by preserving the branching structure of two processes even in the presence of unobservable transitions (that are labeled by a silent action τ); it is required that all intermediate states that are passed through during τ-transitions are related.

A *branching bisimulation* with respect to some *internal symbol* $\tau \in \mathsf{A}$ is a symmetric relation $R \subseteq S \times S$ such that for each $(s,t) \in R$ and each $a \in \mathsf{A}$ the following holds:

- if $s \stackrel{a}{\longrightarrow} s'$, then $t \stackrel{\tau}{\Longrightarrow} t' \stackrel{a}{\longrightarrow} t'' \stackrel{\tau}{\Longrightarrow} t'''$ with $(s,t'),(s',t'') \in R$ for some $t',t'',t''' \in S$.

We write $s \simeq t$ if there is a branching bisimulation R such that $(s,t) \in R$.

Let us introduce the corresponding decision problem: *equivalence checking*.

EQUIVALENCE CHECKING FOR SOME NOTION OF EQUIVALENCE $\equiv \in \{\sim, \approx, \simeq, \ldots\}$

INPUT: A transition system $\mathcal{T}$ and two states s,t of $\mathcal{T}$.
QUESTION: Does $s \equiv t$ hold in $\mathcal{T}$?

We note that sometimes we assume that the input to the equivalence problem consists of two transition systems (from the same class of transition systems) and two of its states, respectively.

Since any class of transition systems that we consider in this thesis is effectively and efficiently closed under disjoint union, we may assume, without loss of generality, that there is only one transition system in the input to the equivalence checking problem (along with two of its states).

Logics

We assume the reader is familiar with first-order logic and monadic second-order logic as well as with temporal logics such as Linear Temporal Logic (LTL), [127], Computation Tree Logic (CTL), as well as the modal μ-calculus [4].

The temporal logics that we investigate in this thesis are all fragments of CTL. We do not include atomic propositions and rather work with transition labels and thus slightly deviate from the classical definition of CTL (which contains atomic propositions but no transition labels). However, since we are only interested in the model checking problem here, one can easily see that model checking formulas of classical CTL can efficiently be reduced to model checking our transition-labeled variant of CTL and vice versa with appropriate adjustments of the input transition systems.

Formulas φ of Computation Tree Logic (CTL) are given by the following grammar, where a ranges over Act:

$$\varphi \quad ::= \quad \texttt{true} \quad | \quad \neg\varphi \quad | \quad \varphi \wedge \varphi \quad | \quad \langle a \rangle \varphi \quad | \quad \mathsf{E}\varphi\mathsf{U}\varphi \quad | \quad \mathsf{E}\varphi\mathsf{WU}\varphi$$

We introduce the abbreviation $\mathsf{EF}\varphi \stackrel{\text{def}}{=} \mathsf{E}\ \texttt{true}\ \mathsf{U}\varphi$. Formulas φ of the *logic* EF are given by the following grammar, where a ranges over Act:

$$\varphi \quad ::= \quad \texttt{true} \quad | \quad \neg\varphi \quad | \quad \varphi \wedge \varphi \quad | \quad \langle a \rangle \varphi \quad | \quad \mathsf{EF}\varphi$$

Formulas of Hennessy-Milner logic (HM) are given by the following grammar, where a ranges over Act:

$$\varphi \quad ::= \quad \texttt{true} \quad | \quad \neg\varphi \quad | \quad \varphi \wedge \varphi \quad | \quad \langle a \rangle \varphi$$

Given a transition system $\mathcal{T} = (S, \mathsf{A}, \{\stackrel{a}{\longrightarrow} |\ a \in \mathsf{A}\})$, some state $s \in S$ of $\mathcal{T}$ and some formula φ, we define $(\mathcal{T}, s) \models \varphi$ by induction on φ as follows, where we recall that $\longrightarrow$ denotes the relation $\left(\bigcup\{\stackrel{a}{\longrightarrow} |\ a \in \mathsf{A}\}\right)^*$:

$$\begin{array}{rcl}
(\mathcal{T}, s) \models \texttt{true} & & \text{for each } s \in S \\
(\mathcal{T}, s) \models \neg\varphi & \stackrel{\text{def}}{\Leftrightarrow} & (\mathcal{T}, s) \not\models \varphi \\
(\mathcal{T}, s) \models \varphi_1 \wedge \varphi_2 & \stackrel{\text{def}}{\Leftrightarrow} & (\mathcal{T}, s) \models \varphi_1 \text{ and } (\mathcal{T}, s) \models \varphi_2 \\
(\mathcal{T}, s) \models \langle a \rangle \varphi & \stackrel{\text{def}}{\Leftrightarrow} & (\mathcal{T}, s') \models \varphi \text{ for some } s' \in S \text{ with } s \stackrel{a}{\longrightarrow} s' \\
(\mathcal{T}, s) \models \mathsf{E}\varphi_1\mathsf{U}\varphi_2 & \stackrel{\text{def}}{\Leftrightarrow} & \exists n \geq 1, s_1, \ldots, s_n \in S : s = s_1 \longrightarrow s_2 \cdots \longrightarrow s_n, \\
 & & \qquad (\mathcal{T}, s_n) \models \varphi_2 \text{ and } \forall i \in [1, n-1] : (\mathcal{T}, s_i) \models \varphi_1 \\
(\mathcal{T}, s) \models \mathsf{E}\varphi_1\mathsf{WU}\varphi_2 & \stackrel{\text{def}}{\Leftrightarrow} & (\mathcal{T}, s) \models \mathsf{E}\varphi_1\mathsf{U}\varphi_2 \text{ or} \\
 & & \exists s_1, s_2, \ldots \in S, : s = s_1, \text{ and } \forall i \geq 1 : (\mathcal{T}, s_i) \models \varphi_1, s_i \longrightarrow s_{i+1}
\end{array}$$

For reasons of simplicity of presentation we sometimes use a variant of the logic EF in this thesis that allows to parametrize the set of action labels in the EF operator. By this we mean formulas of the form $\langle\Gamma^*\rangle\varphi$ for subsets Γ of the action labels that require the transitions of the path to the state satisfying φ all to be labeled by elements of Γ. This parametrized version of EF is slightly more general than the standard definition of EF-logic from above with respect to expressiveness. However, all lower bound results in thesis easily carry over to the restricted definition of EF logic and all upper bounds in this thesis can be proven for the parametrized variant.

Formulas of the parametrized variant of EF over a finite set $\mathsf{A} \subseteq \mathsf{Act}$ of labels are given by the following grammar, where $\Gamma \subseteq \mathsf{A}$:

$$\varphi \quad ::= \quad \texttt{true} \mid \neg\varphi \mid \varphi \wedge \varphi \mid \langle\Gamma\rangle\varphi \mid \langle\Gamma^*\rangle\varphi$$

We write $\langle\Gamma\rangle^n$ (resp. $[\Gamma]^n$) as an abbreviation for a sequence of n consecutive $\langle\Gamma\rangle$'s (resp. $[\Gamma]$'s). For each transition system $\mathcal{T} = (S, \mathsf{A}, \{\stackrel{a}{\longrightarrow} \mid a \in \mathsf{A}\})$ and each formula φ (over A) of parametrized EF we define

$$\begin{array}{rcl} (\mathcal{T}, s) \models \langle\Gamma\rangle\varphi & \stackrel{\text{def}}{\Leftrightarrow} & (\mathcal{T}, s') \models \varphi \text{ for some } s' \in S \text{ with } s \xrightarrow{\Gamma} s' \quad \text{and} \\ (\mathcal{T}, s) \models \langle\Gamma^*\rangle\varphi & \stackrel{\text{def}}{\Leftrightarrow} & (\mathcal{T}, s') \models \varphi \text{ for some } s' \in S \text{ with } s \xrightarrow{\Gamma}{}^{*} s'. \end{array}$$

We define $[\![\varphi]\!]_{\mathcal{T}} \subseteq S$ to be the set of states that satisfy φ.

Let us introduce the *model checking problem*.

MODEL CHECKING FOR A LOGIC $\mathcal{L} \in \{\mathsf{CTL}, \mathsf{EF}, \mathsf{HM}, \ldots\}$

INPUT: A transition system $\mathcal{T} = (S, \mathsf{A}, \{\stackrel{a}{\longrightarrow} \mid a \in \mathsf{A}\})$, a state s of $\mathcal{T}$ and an $\mathcal{L}$-formula φ.

QUESTION: $(\mathcal{T}, s) \models \varphi$?

Following Vardi [191], we distinguish three ways of measuring the computational complexity of the model checking problem: (i) *data complexity* measures the complexity of the model checking problem when the formula is fixed and only the transition system is part of the input, (ii) *expression complexity* measures the complexity relative to a fixed system, thus only the formula is part of the input, and (iii) *combined complexity* assumes that both the system and the formula is part of the input. If not said otherwise, we mean the combined complexity.

3 Infinite-state systems

The study of infinite-state verification has revealed that *unbounded recursions* and *unbounded parallelism* are two of the most important sources of infinity in computer programs. Infinite-state models with unbounded recursions such as Basic Process Algebra (BPA), and Pushdown Systems (PDS) have been studied for a long time (e.g. [6, 146]). The same can be said about infinite-state models with unbounded parallelism, which include Basic Parallel Processes (BPP) and Petri nets (PN), e.g. [45, 86]. While these aforementioned models are either *purely sequential* or *purely parallel*, there are also models that simultaneously inherit both of these features. A well-known example are PA-processes [11], which are a common generalization of BPA and BPP. It is known that all of these models are not Turing-powerful in the sense that decision problems such as reachability is still decidable (e.g. see [30]), which makes them suitable for verification.

In his seminal paper [140], Mayr introduced the Process Rewrite Systems (PRS) hierarchy (see leftmost diagram in Figure 3.2) containing several models of infinite-state systems that generalize the aforementioned well-known models with unbounded recursions and/or unbounded parallelism. The idea is to treat models in the hierarchy as a form of term-rewrite systems, and classify them according to which terms are permitted on the left and right hand side of the rewrite rules. In addition to the aforementioned models of infinite-state systems, the PRS hierarchy contains three new models: (1) Process Rewrite Systems (PRS), which generalize PDS, PA-processes, and Petri nets, (2) PAD-processes, which unify PDS and PA-processes, and (3) PAN-processes, which unify both PA-processes and Petri nets. Mayr showed that the hierarchy is strict with respect to strong bisimulation. Despite of its expressive power PRS is not Turing-powerful since reachability is still decidable for this class.

After having defined Mayr's PRS hierarchy below, we introduce further models of infinite-state systems that are relevant in this thesis and integrate them into the PRS hierarchy with respect to bisimilarity, branching bisimilarity and weak bisimilarity. The aim of this chapter is to mention some relevant results on these classes with respect to the model checking problem and the equivalence checking problem. Since we discuss various different classes of infinite-state systems and the literature on them is large, we do not claim that our list of results is complete. We refer to Mayr's PhD thesis [139] for a more thorough overview of model checking classes of infinite-state systems in the PRS hierarchy against various temporal logics.

The technical contribution of this chapter is discussed in Section 3.1, where we integrate ground tree rewrite systems into Mayr's PRS hierarchy with respect to bisimilarity, branching bisimilarity and weak bisimilarity.

In the following, let us fix a countable set of process constants (a.k.a. process variables) $\mathcal{X} = \{A, B, C, D, \ldots\}$. The set of *process terms* t is given by the following grammar, where X ranges over $\mathcal{X}$:

$$t \quad ::= \quad 0 \quad | \quad X \quad | \quad t.t \quad | \quad t \| t$$

The *size* $\mathsf{size}(t)$ of a term t is inductively defined as $\mathsf{size}(0) = \mathsf{size}(X) = 1$ and $\mathsf{size}(t_1.t_2) = \mathsf{size}(t_1 \| t_2) = \mathsf{size}(t_1) + \mathsf{size}(t_2) + 1$. The operator '.' is said to be *sequential composition* and assumed to be associative, while the operator $\|$ is referred to as *parallel composition* is assumed both associative and commutative. The smallest equivalence relation on terms that is associative and commutative for parallel composition and that is associative for sequential composition is denoted by $\equiv$ in the following. Mayr [140] distinguishes the following classes of process terms:

- $\mathbb{1}$ Terms consisting of a single constant $X \in \mathcal{X}$.
- $\mathbb{S}$ Process terms without any occurrence of parallel composition.
- $\mathbb{P}$ Process terms without any occurrence of sequential composition.
- $\mathbb{G}$ Arbitrary process terms possibly with sequential or parallel compositions.

A *process rewrite system (*PRS*)* is a tuple $\mathcal{P} = (\Sigma, \mathsf{A}, \Delta)$, where

- $\Sigma \subseteq \mathcal{X}$ is a finite set of process constants,
- $\mathsf{A} \subseteq \mathsf{Act}$ is a finite set of atomic actions, and
- Δ is a finite set of rewrite rules of the form $t_1 \mapsto_a t_2$, where t_1 and t_2 are terms over the process constants in Σ with $t_1 \not\equiv 0$ and $a \in \mathsf{A}$.

Let us discuss the underlying transition system $\mathcal{T}(\mathcal{P}) = (S, \mathsf{A}, \{\stackrel{a}{\longrightarrow} |\ a \in \mathsf{A}\})$. The state set S is defined to be the set of all equivalence classes of all terms built over Σ modulo the above-mentioned equivalence relation $\equiv$. Moreover, for each $a \in \mathsf{A}$, the transition relation $\stackrel{a}{\longrightarrow}$ is implicity defined by the following inference rules:

$$\frac{t_1 \mapsto_a t_2 \in \Delta}{t_1 \stackrel{a}{\longrightarrow} t_2} \qquad \frac{t_1 \stackrel{a}{\longrightarrow} t_1'}{t_1 \| t_2 \stackrel{a}{\longrightarrow} t_1' \| t_2} \qquad \frac{t_1 \stackrel{a}{\longrightarrow} t_1'}{t_1.t_2 \stackrel{a}{\longrightarrow} t_1'.t_2}$$

Other models in in the PRS hierarchy are *Finite Systems* (FIN), *Basic Process Algebra* (BPA), *Basic Parallel Processes* (BPP), *Pushdown Systems* (PDS), *Petri nets* (PN), *PA processes* (PA), *PAD processes* (PAD), and *PAN processes* (PAN). They can be defined by restricting the terms that are allowed on the left hand side ℓ and on the right hand side r of the PRS rewrite rules and are abbreviated by PRS(ℓ, r), where $\ell, r \in \{\mathbb{1}, \mathbb{S}, \mathbb{P}, \mathbb{G}\}$. An important result by Mayr [140] is that the PRS hierarchy is strict with respect to strong bisimulation .

- *Finite Systems* $=$ FIN $=$ *PRS*$(\mathbb{1}, \mathbb{1})$. It is easy to see that the class of finite transition systems coincides with PRS$(\mathbb{1}, \mathbb{1})$: there is a one-to-one correspondence between the process constants and the states of the underlying transition system that it describes as well as a one-to-one correspondence between the rewrite rules and the transitions in the underlying finite system.

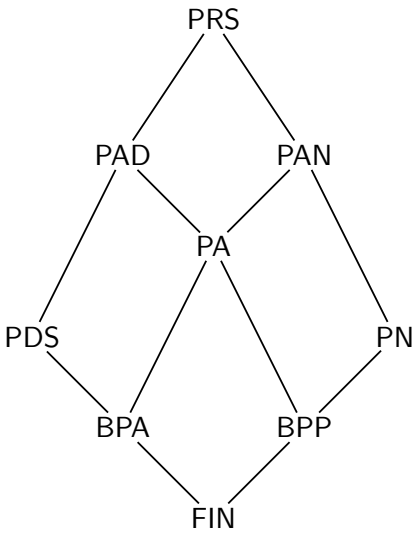


Figure 3.1: Mayr's PRS hierarchy, strictness is shown with respect to bisimulation equivalence [140].

Model checking. The computational complexity of model checking on finite systems is very well understood, model checking LTL, first-order and monadic second-order logic are PSPACE-complete, where P-completeness holds for CTL modal logic, EF logic, and HM [123] and reachability is NL-complete, just to mention few results.

Model checking the modal μ-calculus, which is polynomial time equivalent to determining the winner of a parity game [59], takes a prominent exception and is undoubtedly one of the biggest open problems in logic in computer science: The best-known upper bound is UP $\cap$ co-UP [111] (see also for fast implementations [161, 112, 67]) and the best-known lower bound is P.

Equivalence Checking. By the classical partition refinement algorithm, one can decide bisimulation equivalence of finite systems in polynomial time [150, 113]. Hardness for P has been shown in [7]. The same complexity bounds hold for weak bisimilarity. On the other hand, trace equivalence on finite systems is PSPACE-complete since it is polynomial time equivalent to the equivalence problem of finite word automata.

- *Pushdown Systems* $=$ PDS $=$ *PRS*$(\mathbb{S}, \mathbb{S})$. Pushdown systems can equivalently be defined as the configuration graphs of pushdown automata without ε-transitions. Speaking in terms of Mayr's PRS hierarchy, they make up the graphs that one obtains from applying purely sequential rewriting (i.e. both on the left-hand side and right-hand side of rewrite rules the terms are purely sequential). In recent years a lot of research has been devoted to verification of pushdown systems. A central reason for this interest is surely that pushdown systems allow to abstractly mimic the call and return behavior of procedural programs. Let us mention an equivalent way of defining pushdown systems. Alternatively a pushdown system can be seen to be given by a tuple $\mathcal{P} = (Q, \mathsf{A}, \Gamma, \{\overset{a}{\hookrightarrow} |\ a \in \mathsf{A}\})$, where Q is a finite set of *control states*, $\mathsf{A} \subseteq$ Act is a finite set of action labels, Γ is a

finite *stack alphabet* that contains a distinguished *bottom-of-stack symbol* $\bot$, and where finally for each $a \in \mathsf{A}$ we have that $\overset{a}{\hookrightarrow}$ is a set of rewrite rules of the form $pX \overset{a}{\hookrightarrow} qw$, where $p, q \in Q$ and $X \in \Gamma, w \in \Gamma^*$ satisfying $X = \bot$ implies $w \in \Gamma^*\bot$ and $X \neq \bot$ implies $w \in \Gamma^*$. The transition system $\mathcal{T}(\mathcal{P})$ is defined as $(S, \mathsf{A}, \{\overset{a}{\longrightarrow} | \ a \in \mathsf{A}\})$, where $S = Q(\Gamma \setminus \{\bot\})^*\bot$ and where each rule of the kind $p\bot \overset{a}{\hookrightarrow} qw$ induces the transition $p\bot \overset{a}{\longrightarrow} qw$ and each rule $pX \overset{q}{\hookrightarrow} w$ with $X \neq \bot$ induces the transitions $pXz \overset{a}{\longrightarrow} qwz$ for each $z \in (\Gamma \setminus \{\bot\})^*\bot$.

In this thesis we prefer to use the latter notion, in particular when defining one-counter systems below.

Model Checking. By the use of automata-theoretic constructions it is shown that for each regular set of configurations the set of reachable configurations (resp. the set of configurations that can reach this set) is effectively regular again and computable in polynomial time [29, 18]. In [18] it has been shown that such effective automata-theoretic constructions can be used to show that model checking pushdown systems with respect to various logics such as the alternation-free μ-calculus or LTL is decidable and moreover yield exact complexity bounds, we also refer to [118]. Rabin's theorem states that the monadic-second order (MSO) theory of the infinite binary tree is decidable. Since the each pushdown system is MSO-interpretable in the infinite binary tree it follows that MSO model checking for pushdown systems is decidable, however with nonelementary complexity [180]. Two further milestone results in this area were proven by Walukiwicz on the one hand, who proved that model checking the μ-calculus is in fact elementary and EXP-complete [197], and by Kupferman and Vardi, on the other hand, who have used the automata-theoretic approach to obtain this EXP upper bound [120]. The same complexity already holds for the fragment CTL of the modal μ-calculus [196], whereas model checking its fragment EF is PSPACE-complete [18, 196]; PSPACE-hardness already holds for HM [139]. Model checking LTL on pushdown graphs is complete for EXP [18]. Model checking CTL^* on pushdown systems has been shown 2EXP-complete in [23]. Finally, model checking PDL on pushdown systems has been studied systematically [72], where it has also been proven that model checking PDL with intersection and converse is 2EXP-complete.

Equivalence Checking. A folklore result is that language equivalence (in fact already universality) of pushdown automata is undecidable. However, a celebrated result due to Sénizergues states that language equivalence of deterministic pushdown automata (even with ε-transitions) is decidable [165]. Later Stirling proved that the problem is in fact primitive recursive [178]. Sénizergues later proved that bisimulation equivalence of pushdown systems (where possibly ε-transitions occur but only in a deterministic popping fashion) is decidable [167]. In fact, in [167] it is even shown that decidability still holds for equational graphs of finite out-degree. Unfortunately, to date there is no complexity-theoretic upper bound known for this problem. If one allows ε-transitions, then *weak* bisimilarity between is undecidable already for one-counter systems, i.e. the configuration graphs of pushdown automata over a singleton stack alphabet, by [142].

In Section 8.2 we discuss a nonelementary lower bound for bisimilarity on pushdown systems – it improves the previously best known EXP lower bound by Kučera and Mayr

for this problem [121].

- *Basic Process Algebras* $=$ BPA $= \mathit{PRS}(\mathbb{1}, \mathbb{S})$. The transition graphs defined by Basic Process Algebras can equivalently be viewed as the transition graph of pushdown automata, where the set of control states is a singleton. It is well-known that with respect to trace equivalence BPA and PDS are equivalent. More precisely, for each pushdown system $\mathcal{P}$ and configuration c of $\mathcal{P}$ there exists a BPA $\mathcal{P}'$ along with a configuration c' of $\mathcal{P}'$ such that c and c' are trace equivalent.

  Model Checking. Since the infinite binary tree can still be described by a BPA the MSO of BPA's is generally not elementary. The computational complexity of model checking the modal μ-calculus, CTL, EF, LTL or the linear-time μ-calculus on BPA coincides with the respective complexity on pushdown systems, ranging between PSPACE-completeness and EXP-completeness. However, in stark contrast to model checking pushdown systems, the data complexity all of the latter problems is known to be solvable in polynomial time, cf. [139].

  Equivalence Checking. Since BPA are trace equivalent to PDS trace equivalence is undecidable. Burkart, Caucal and Steffen proved that bisimilarity of BPA is decidable in doubly exponential time [31]. Recently Jančar presented a simplified proof of the 2EXP upper bound [102]. Also the lower bound has recently been lifted from PSPACE to EXP by Kiefer [114]. Thus, there is still an exponential complexity gap for this problem. However, when normed basic processes algebras are considered bisimilarity can be solved in polynomial time [53]. For weak bisimilarity of BPA the situation is less clear: the problem is not known to be decidable. Branching bisimilarity of normed basic process algebras has recently been announced as decidable by [68], see whereas a more restricted variant of normedness has been proven to be decidable in Σ_2^p by Caucal, Huynh and Tran [42].

- *Petri Nets*$=$ PN $= \mathit{PRS}(\mathbb{P}, \mathbb{P})$. *Petri nets* or *Vector Addition Systems* are a well-studied infite-state model for modeling concurrency. As presented in Mayr's framework pushdown systems are obtained from purely sequential rewriting, whereas Petri nets make up the purely parallel counter part.

  Model Checking. A classical result by Kosarju [117] and Mayr [136] states that reachability of Petri nets is decidable. Only recently a significantly simplified proof has been presented by Leroux [125]. This problem is of particular interest not only because its complexity is far from being well understood; the best-known upper bound that one can derive from the above-mentioned papers is a non-primitive recursive upper one, whereas an EXPSPACE lower bound already prevails since over thirty years [37]. Further classical problems on Petri nets include the coverability problem, the boundedness problem and the language regularity problem, which all have been shown to be decidable in EXPSPACE [58, 157].

  Already model checking of CTL's fragment EF is undecidable over Petri nets but at least as hard as the reachability problem [60, 61]. However, model checking LTL and the linear-time μ-calculus [60, 85] is decidable.

  Equivalence Checking. Trace equivalence of Petri nets has been shown undecidable by

Hirshfeld; the lower bound already holds for BPP [92]. Undecidability of bisimilarity of Petri nets has been established by Jančar [99]. However decidability is known for the problem of deciding if a given Petri net is bisimilar to a finite system or to decide of a given Petri net is bisimilar to some finite system [105]. Model checking first-order fragments on the reachability graphs of Petri nets has recently been studied [55]. Finally, model checking the coverability graph of Petri nets against CTL variants has recently been investigated [13].

- *Basic Parallel Processes*$=$ BPP $=$ *PRS*$(\mathbb{1}, \mathbb{P})$. Basic Parallel Processes are also known as communication-free Petri nets, i.e. Petri nets, where each transition consumes exactly one token. This class has been introduced in [45].

  Model Checking. Reachability for BPP has been proven NP-complete by Esparza [61]. Decidability of model checking LTL carries over from Petri nets. It is shown in [139] that model checking LTL over BPP is at least as hard as reachability for Petri nets. Recall that model checking EF is undecidable for Petri nets. Mayr showed that model checking EF on BPP is in fact PSPACE-complete [137]. However, already model checking CTL's fragment EG (a formula $\mathsf{EG}\varphi$ holds in a state s if from s there exists an infinite path whose states all satisfy φ) is undecidable on BPP as shown by Esparza and Kiehn [62].

  Equivalence Checking. Jančar proved that bisimilarity of Petri nets is undecidable. He showed a couple of years later that bisimilarity of BPP is in fact decidable and in PSPACE [101] matching the PSPACE lower bound proven by Srba [172]. Thus, BPA concists one of few classes of infinite-state systems, where bisimilarity is known to be both decidable and its precise complexity known. Recent progress has been provided by Czerwinski, Hofman and Lasota who proved that branching bisimilarity between normed basic parallel processes is decidable [52]. To date, it is a major open problem whether weak bisimilarity of basic parallel processes is decidable, we refer to [96] for a recent devolopment.

- *PA processes*$=$ PA $=$ *PRS*$(\mathbb{1}, \mathbb{G})$ *and PAD processes*$=$ PAD $=$ *PRS*$(\mathbb{S}, \mathbb{G})$. PAD processes can be used to model systems that behave nondeterminstically, concurrently (without communication) and have the possibility to call subroutines whose return value can be taken into account. PA processes can be seen as the join of BPA and BPP: the left-hand side of any rewrite rule may only consist of one symbol whereas the right-hand side my be any term: Thus, they do not posses any means of passing information of concurrently running processes nor can they take the return value of subroutines into account.

  Model Checking. Bouajjani and Habermehl proved that model checking LTL over PA processes is undecidable [19]. Undecidability of CTL is inherited from the undecidability of BPP. Lugiez and Schnoebelen proved that model checking first-order logic with reachability is decidable over PA [130]. Model checking EF on PAD was shown to be decidable by Mayr [138]. The upper proof [138] provides a sophisticated procedure running nonelementary in the input formula. It is left as an open problem whether this huge complexity is inherent.

  In this thesis we contribute to model checking PAD in two ways. Firstly, we state that given a PAD process one can effectively construct a ground tree rewrite system (to be

defined below) that is branching bisimilar (a notion of equivalence that lies between bisimilarity and weak bisimilarity) to it. By employing well-known algorithms for model checking EF on ground tree rewrite systems [128], we obtain an alternative proof for the decidability of model checking EF on PAD. Secondly, we answer the above-mentioned complexity question affirmatively: We prove a nonelementary lower bound already for model checking two concurrent basic process algebras in Chapter 7.

Equivalence Checking. Hirshfeld and Jerrum showed that bisimilarity of normed PA processes is decidable in nondeterministic doubly exponential time [93]. To date it is unknown whether bisimilarity of general PA processes is decidable. Yet, weak bisimilarity of PA processes is highly undecidable [173, 110].

- *PRS-processes*$=$ PRS $=$ *PRS*$(\mathbb{G}, \mathbb{G})$ and *PAN processes*$=$ PAN $=$ *PRS*$(\mathbb{P}, \mathbb{G})$. The most general variant PRS are systems obtained by ground term rewriting without any restrictions of the terms on the left-hand side or right-hand side of the rules. PAN processes restrict the left-hand side of any rule to consist of parallel terms only. We summarize these two systems since the decidability/complexity status both for model checking and for equivalence checking is the same for the two models.

  Model Checking. Mayr proved that reachability is decidable on PRS and even deciding if a PRS process can reach another process [139] that can execute a certain (definable) set of transitions. It is not hard to see that model checking HM is decidable on PRS since the out-degree of every PRS process is finite. On the other hand, undecidability already holds for model checking LTL and EF for PAN [139].

  Equivalence Checking. Trace equivalence, bisimilarity and thus weak bisimilarity are all undecidable on PRS – undecidability for all these problems is inherited from Petri nets.

3.1 Ground tree rewrite systems and its integration into Mayr's PRS hierarchy

Before the PRS hierarchy was introduced, another class of infinite-state systems called Ground Tree/Term Rewrite Systems (GTRS) already emerged in the term rewriting community as a class with good decidability properties. Recall that pushdown systems can be seen as systems whose nodes are essentially finite words and whose transitions are given by a finite set of word rewriting rules that are applied in a prefix-rewriting fashion. Ground tree rewrite systems rewrite systems can be seen as the generalization of the latter to rewriting finite ranked trees.

Model Checking. While extending the expressive power of PDS, GTRS still enjoys decidability of reachability (e.g. [25, 50]), recurrent reachability [128], model checking first-order logic with reachability [56, 48], and model checking the fragments LTL_{det} and $\mathsf{LTL}(\mathbf{F}_{\mathrm{s}}, \mathbf{G}_{\mathrm{s}})$ of LTL [185, 184]. Due to the tree structures that GTRS use in their rewrite rules, GTRS can be used to model concurrent systems with both unbounded parallelism (a new thread may be spawned at any given time) and unbounded recursions (each thread may behave as a pushdown system). When comparing the definitions of PRS (and subclasses thereof) and GTRS, one cannot help

but notice their similarity. Moreover, there is a striking similarity between the problems that are decidable (and undecidable) over subclasses of PRS like PA/PAD processes and GTRS. For example, reachability, EF model checking, and $\mathsf{LTL}(\mathbf{F}_s, \mathbf{G}_s)$ and LTL_{det} model checking are decidable for both PAD-processes and GTRS [24, 128, 140, 141, 184, 185]. Furthermore, model checking general LTL properties is undecidable for both PA-processes and GTRS [24, 185].

In Section 9 we concern ourselves with model checking (regular) ground tree rewrite systems against specification in the logic EF and prove that already model checking EF formulas with at most two nesting of the EF operator are hard to model check on GTRS: the problem is nonelementary. We provide a $\mathsf{P}^{\mathsf{NEXP}}$-completeness result for model checking EF formulas with at most one nesting of the EF operator.

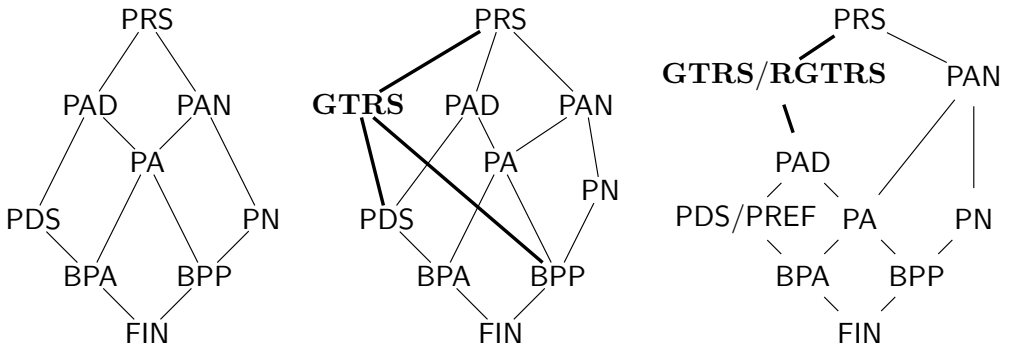
Equivalence Checking. Undecidability of trace equivalence and of weak bisimilarity on ground tree rewrite systems is inherited from pushdown systems [110]. It is an interesting open problem whether bisimilarity of GTRS is decidable. In Chapter 9 we provide a coNEXP upper bound of deciding bisimilarity between a ground tree rewrite system and a finite system. Moreover, we show that deciding if a given regular ground tree rewrite is bisimilar to a given finite transition system is nonelementary.

Despite the above-mentioned similarities between PA/PAD and GTRS, the precise connection between the PRS hierarchy and GTRS has not been investigated until recently. A particular technical difference between the classes PA/PAD and GTRS is that the states of the underlying transition system for PA/PAD is defined modulo the equivalence $\equiv$ on terms, whereas the states of a GTRS are indeed finite ranked trees themselves (and not the equivalence classes on them).

For the rest of this section we discuss a joint work [78] with Anthony Widjaja Lin, which has appeared as a conference paper in [75]. We extend Mayr's PRS hierarchy by integrating Ground Tree Rewrite Systems. We pinpoint the precise connection between the expressive powers of GTRS and models inside the PRS hierarchy with respect to strong, branching, and weak bisimulation equivalence.

The results are summarized in the middle and right diagrams in Figure 3.2.

Our investigation is inspired by the work of Lugiez and Schnoebelen [131] and Bouajjani and Touili [21], which study PRS (or subclasses thereof) by first distinguishing process terms that are "equivalent" in Mayr's sense [140]. This approach allows them to make use of techniques from classical theory of tree automata for solving interesting problems over PRS (or subclasses thereof). Our translation from PAD to GTRS is similar in spirit. We also show that Regular Ground Tree Rewrite Systems (RGTRS) [128] — the extension of GTRS with possibly infinitely many GTRS rules compactly represented as tree automata — have the same expressive power as GTRS up to branching/weak bisimulation. Along the same ideas that are used in the latter proof one can show that PDS is equivalent to prefix-recognizable systems , abbreviated as PREF, (cf. see [30]) up to branching/weak bisimulation. On the other hand, when we investigate the expressive power of GTRS with respect to strong bisimulation, we found that PAD (even PA) is no longer subsumed in GTRS. Despite this, we can show that up to strong bisimulation GTRS is strictly more expressive than BPP and PDS, and is strictly subsumed in PRS. Finally, we mention that our results imply that Mayr's PRS hierarchy is also strict with respect to weak bisimulation equivalence.



Mayr's original PRS hierarchy w.r.t. strong bisimulation $\sim$

Our refinement with GTRS w.r.t. strong bisimulation $\sim$

Our refinement with GTRS w.r.t. branching bisimulation $\simeq$ and weak bisimulation $\approx$

Figure 3.2: Depictions of Mayr's PRS hierarchy and their refinements via GTRS as Hasse diagrams (the top being the most expressive). The leftmost diagram is the original (strict) PRS hierarchy where expressiveness is measured with respect to strong bisimulation. The middle (resp. right) diagram is a strict refinement via GTRS with respect to strong (resp. weak/branching) bisimulation.

There are other models of multithreaded programs with unbounded recursions that have been studied in the literature. Specifically, we mention Dynamic Pushdown Networks (DPN) and extensions thereof (e.g. see [20]) since an extension of DPN given in [20] also extends PAD-processes. We leave it for future work to study the precise connections between these models and GTRS.

Let us formally introduce regular ground tree rewrite systems (RGTRS), ground tree rewrite systems (GTRS) and prefix-recognizable systems (PREF).

Let us first define ranked trees. Let $\preceq$ denote the prefix order on $\mathbb{N}^*$, i.e. $x \preceq y$ for $x, y \in \mathbb{N}^*$ if there is some $z \in \mathbb{N}^*$ such that $y = xz$, and $x \prec y$ if $x \preceq y$ and $x \neq y$. A *ranked alphabet* is a collection of finite and pairwise disjoint alphabets $\Sigma = (\Sigma_i)_{i\in[0,k]}$ for some $k \geq 0$. For simplicity we identify Σ with $\bigcup_{i\in[0,k]} \Sigma_i$. A *ranked tree* (over the ranked alphabet Σ) is a mapping $T : D_T \to \Sigma$, where $D_T \subseteq [1,k]^*$ satisfies the following: D_T is non-empty, finite and prefix-closed and for each $x \in D_T$ with $T(x) \in \Sigma_i$ we have $x1, \ldots, xi \in D_T$ and $xj \notin D_T$ for each $j > i$. We say that D_T is the *domain* of T — we call these elements *nodes*. A *leaf* is a node x with $T(x) \in \Sigma_0$. We also refer to $\varepsilon \in D_T$ as the *root* of T. By Trees_Σ we denote the set of all ranked trees over the ranked alphabet Σ. We also use the usual term representation of trees, e.g. if T is a tree with root a and left (resp. right) subtree T_1 (resp. T_2) we have $T = a(T_1, T_2)$.

Let T be a ranked tree and let x be a node of T. We define $xD_T = \{xy \in [1,k]^* \mid y \in D_T\}$ and $x^{-1}D_T = \{y \in [1,k]^* \mid xy \in D_T\}$. By $T^{\downarrow x}$ we denote the *subtree of* T with root x, i.e. the tree with domain $D_{T^{\downarrow x}} = x^{-1}D_T$ defined as $T^{\downarrow x}(y) = T(xy)$. Let $T, T' \in \mathsf{Trees}_\Sigma$ and let x be a node of T. We define $T[x/T']$ to be the tree that is obtained by replacing $T^{\downarrow x}$ in T by T';

more formally $D_{T[x/T']} = (D_T \setminus xD_{T\downarrow x}) \cup xD_{T'}$ with $T[x/T'](y) = T(y)$ if $y \in D_T \setminus xD_{T\downarrow x}$ and $T[x/T'](y) = T'(z)$ if $y = xz$ with $z \in D_{T'}$. Define $|T| = |D_T|$ as the number of nodes in a tree T.

A *regular ground tree rewrite system* (RGTRS) is a tuple $\mathcal{R} = (\Sigma, \mathsf{A}, R)$, where Σ is a ranked alphabet, $\mathsf{A} \subseteq \mathsf{Act}$ is a finite set of action labels and where R is finite set of rewrite rules $L \overset{a}{\hookrightarrow} L'$, where L and L' are regular tree languages given as nondeterministic bottom-up tree automata (cf. [49] for more details). The transition system defined by $\mathcal{R}$ is $\mathcal{T}(\mathcal{R}) = (\mathsf{Trees}_\Sigma, \mathsf{A}, \{\overset{a}{\longrightarrow} | a \in \mathsf{A}\})$, where for each $a \in \mathsf{A}$, we have $T \overset{a}{\longrightarrow} T'$ if and only if there is some $x \in D_T$ and some rule $L \overset{a}{\hookrightarrow} L' \in R$ such that $T^{\downarrow x} \in L$ and $T' = T[x/T'']$ for some $T'' \in L'$ (we say that the rule was applied at node x).

A *ground tree rewrite system* (GTRS) is an RGTRS $\mathcal{R} = (\Sigma, \mathsf{A}, R)$, where for each $L \overset{a}{\hookrightarrow} L' \in R$ we have that both $L = \{T\}$ and $L' = \{T'\}$ is a singleton; we also write $T \overset{a}{\hookrightarrow} T' \in R$ for this.

A *prefix-recognizable system* (PREF) is an RGTRS $\mathcal{R} = (\Sigma, \mathsf{A}, R)$, where only Σ_0 and Σ_1 may be non-empty. We note that analogously pushdown systems are precisely those GTRS $\mathcal{R} = (\Sigma, \mathsf{A}, R)$, where only A_0 and A_1 may be non-empty.

While it follows from known results that there is a Petri net that is not trace equivalent to any GTRS our first main result states that the expressive power of GTRS with respect to branching and weak bisimulation is strictly above PAD.

Theorem 3.1 ([78]) *Given a state s of some PAD $\mathcal{P}$ one can compute in polynomial time a GTRS $\mathcal{R}$ and a tree (state) T of $\mathcal{T}(\mathcal{R})$ such that $s \simeq T$.* □

This result allows us to transfer some decidability/complexity results of model checking problems over GTRS to PA and PAD processes. In particular, it gives a simple proof of the decidability of the problem of model checking the logic EF over PAD [141], and decidability (with better complexity upper bounds that we will not state explicitly here) of the problem of model checking the fragments LTL_{det} and $\mathsf{LTL}(\mathbf{F}_\mathrm{s}, \mathbf{G}_\mathrm{s})$ of LTL over PAD (this decidability result was initially given in [24] without upper bounds). Since in [56] it has been shown that model checking first-order logic with reachability is decidable over GTRS, we obtain as a corollary that model checking the logics EF, and LTL's fragments LTL_{det} and $\mathsf{LTL}(\mathbf{F}_\mathrm{s}, \mathbf{G}_\mathrm{s})$ (see [185, 184] for further details) are all decidable over PAD.

Corollary 3.2 ([78]) *Model checking any of the logics EF, LTL_{det} and $\mathsf{LTL}(\boldsymbol{F}_s, \boldsymbol{G}_s)$ is decidable over PAD.* □

Since every GTRS is of course an RGTRS and every PDS is of course a PREF, the below-stated Theorem 3.3 allows us to deduce that RGTRS and GTRS are equivalent up to branching bisimulation and the same holds for PREF and PDS. Although the proof of Theorem 3.3 is not very complicated, the reason why this could be of interest is for instance that previously both bisimilarity of regular ground tree rewrite systems against finite systems and weak bisimilarity of ground tree rewrite systems against finite systems have been studied [76] separately. Similar remarks apply to bisimilarity of prefix-recognizable systems and weak bisimilarity between pushdown systems [110]. Theorem 3.3 states that both equivalence checking problems are in fact equivalent up to polynomial time reductions.

Theorem 3.3 ([78]) *The following containments hold:*

1. *Given a state T of some given* RGTRS *one can construct in polynomial time some state T' of some* GTRS $\mathcal{P}$ *such that $T \simeq T'$.*
2. *Given a state T of some given* PREF *one can construct in polynomial time some state T' of some* PDS $\mathcal{P}'$ *such that $T \simeq T'$.*
3. *Given a state s of some given* BPP *one can construct in polynomial time some state T of* GTRS $\mathcal{P}$ *such that $T \sim s$.* □

We have obtained the following separation results, whose proofs make use of established automata-theoretic techniques in a sophisticated way. We do not discuss the proof ideas here.

Theorem 3.4 ([78]) *The following strictness results hold:*

1. *There exists a state s of a* PA *such that no state of any* GTRS *is bisimilar to s.*
2. *There exists a state s of a* GTRS *such that no state of any* PAD *is weakly bisimilar to s.*
3. *There exists state s of a* PDS *such that no state of any* PAN *is weakly bisimilar to s.* □

3.2 One-counter systems

A *one-counter system* is a configuration graph of a pushdown automaton $\mathcal{P} = (Q, \mathsf{A}, \Gamma, \{\overset{a}{\hookrightarrow} | a \in \mathsf{A}\})$ that satisfies $\Gamma = \{A, \bot\}$ for some symbol A: thus apart from the bottom-of-stack symbol $\bot$ there is exactly one further stack symbol A. It is more convenient to abbreviate states (configurations) $(q, A^n\bot)$ in $\mathcal{T}(\mathcal{P})$ by $q(n)$. Too, it is more convenient to write a one-counter system as a tuple $\mathcal{P} = (Q, \mathsf{A}, \delta_0, \delta_{>0})$, where $\delta_0 \subseteq Q \times \mathsf{A} \times \{0, +1\} \times Q$ and where $\delta_{>0} \subseteq Q \times \mathsf{A} \times \{-1, 0, +1\} \times Q$ with the obvious meaning; e.g. an element $(q, a, -1, q') \in \delta_{>0}$ would allow, in case the current counter is positive, to change from q to q' on reading the letter a and hereby decreasing the counter by one. Hence, one-counter systems can be seen as one of the simplest means to model infinite-state systems – they can be seen to consist of a special finite transition system corresponding to counter value 0 that is connected to ω copies of some finite transition system corresponding to the positive counter values. It is folklore how one-counter systems integrate to Mayr's PRS hierarchy with respect to bisimilarity and branching and weak bisimilarity, cf. Figure 3.3.

Let us briefly discuss the extensions of succinct and parametric one-counter systems without providing rigorous definitions. *Succinct one-counter systems* are one-counter systems in which the increments and decrements that appear in the rewrite rules are specified by numbers *given in binary*. *Parametric one-counter systems* "generalize" the latter by allowing in the rewrite rule to increment the counter by the value of a variable (ranging over a set of variables X) that can be instantiated by any integer. For parametric one-counter systems it remains to discuss how the model checking problem is defined. Given a parametric one-counter automaton $\mathcal{P}$ and a configuration $q(n)$, we write $(\mathcal{T}(\mathcal{P}), q(n)) \models \varphi$ if for *every* assignment $\alpha : X \to \mathbb{Z}$ of the

parameters that occur on transitions of $\mathcal{P}$ we have that the succinct one-counter system $\mathcal{T}_\alpha$ that is induced by α satisfies $(\mathcal{T}_\alpha, q(n)) \models \varphi$. Analogously, the reachability problem in a given parametric one-counter system asks whether reachability holds in *some* succinct one-counter system that one obtains in some way by evaluating the parameters.

Model Checking. A folklore result states that reachability of one-counter systems is NL-complete. The reachability problem for succinct and parametric one-counter systems has recently shown to be NP-complete [83]. It is also easy to see that model checking HM on one-counter systems is P-complete. While model checking first-order logic with reachability is nonelementary already over the complete binary tree (which is in fact a BPA but not a OCS), To recently proved that its complexity drops to PSPACE when model checking (even asynchronous products of) one-counter systems [182]. To's proof is inspired by an upper bound technique that we have developed for model checking EF on one-counter systems (Chapter 6).

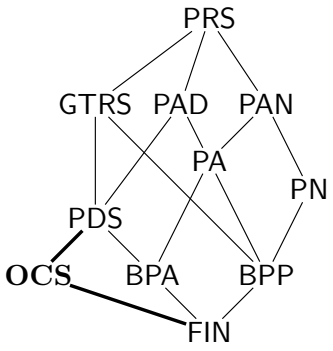
Serre proved that the computational complexity of model checking the modal μ-calculus on OCS is in PSPACE [169] and thus simpler than on PDS (unless PSPACE $=$ EXP), where it is already EXP-complete for CTL [119, 196, 198], as mentioned above. Since the emptiness of alternating finite word automaton over a unary alphabet [98] can easily be expressed by a μ-calculus formula, a matching PSPACE lower bound for model checking the modal μ-calculus on OCS follows, in fact already for a fixed formula.

In Section 5 we analyze the computational complexity of model checking CTL. We prove that model checking CTL on any fixed one-counter system can be done in polynomial time provided the input CTL formula has only constantly many leftward nestings of the until operator (a notion that we make more precise in Chapter 5). Concerning lower bounds, we show that both the expression complexity and the data complexity of model checking CTL on OCS is PSPACE-hard. In particular for hardness of the data complexity we develop new lower bound techniques inspired from two deep results in complexity theory. Too, we discucss that for succinct one-counter systems the data complexity of model checking CTL is EXPSPACE-complete.

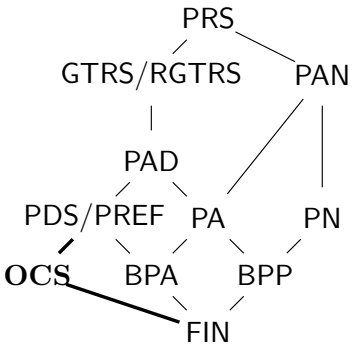
The computational complexity of model checking EF on OCS has been shown to be DP-hard [108], where DP consists of all languages that are the intersection of a language in NP and a language in coNP. This lower bound has slightly been improved to $\mathsf{P}_{\|}^{\mathsf{NP}}$-hardness [72], where $\mathsf{P}_{\|}^{\mathsf{NP}}$ is the set of all problems that can be solved by a deterministic polynomial time bounded Turing machine that has *parallel* access to an oracle from NP or equivalently by a deterministic polynomial time bounded Turing machine that has access to an oracle from NP but queries the oracle only logarithmically many times [194, 171], see also [162] for a finer analysis. In Chapter 5 we provide a P^{NP} lower bound for this problem, by making use of our lower bound technique. A matching P^{NP} upper bound for model checking EF on one-counter systems is content of Chapter 6. We obtain as a corollary that it is P^{NP}-complete to decide whether a given one-counter system is weakly bisimilar to a given finite transition system.

The complexity of model checking succinct one-counter systems is proven PSPACE-complete for the logics EF and HM in Chapter 6. We show that even model checking HM on parametric one-counter systems is PSPACE-complete, whereas it becomes undecidable for model checking EF.

Equivalence Checking. It is a folklore result that trace equivalence of one-counter systems is



w.r.t. strong bisimulation $\sim$



w.r.t. branching bisimulation $\simeq$
and weak bisimulation $\approx$

Figure 3.3: Refinement of Mayr's PRS hierarchy with GTRS and OCS as Hasse diagrams (the top being the most expressive).

undecidable. Decidability of bisimilarity of OCS follows from its decidability for PDS [167]. However, Jančar independently proved decidability of bisimilarity on OCS [100], however the proof from [100] only allowed to prove an elementary upper bound: in fact Yen [200] analyzed Jančar's algorithm and proved that it runs in triply exponential space. The previously best known lower bound was first DP [108] but then lifted to PSPACE-hardness by Srba [175]. We discuss in Chapter 4 our result that bisimilarity of OCS is in fact PSPACE-complete. We also prove that one can decide in polynomial time whether for a given one-counter system there exists some finite system that is bisimilar to it. Mayr proved that weak bisimilarity on one-counter systems is undecidable [142]. It is not hard to see that the question whether a given one-counter system can be simulated by another one is also undecidable. However, when they both cannot test for zero, then the latter problem becomes decidable [1, 109]. In fact, very recently it has been shown that even weak similarity for one-counter systems without zero-test is decidable [95], which is surprising since weak bisimilarity is undecidable for them [142].

Trace equivalence of deterministic one-counter automata (which are deterministic pushdown automata over a singleton stack alphabet plus a bottom-of-stack symbol and possibly containing ε-transitions) has been proven decidable by Valiant and Paterson [186] many years before the decidability for deterministic pushdown automata has been proven by Sénizergues [164]. The running time of the algorithm in [186] is $2^{O(\sqrt{n \log n})}$. By a simple analysis of their proof, a PSPACE upper bound can be derived for this problem. It follows immediately from digraph reachability that this problem is NL-hard. Thus, there has been still an exponential complexity gap between NL and PSPACE for trace equivalence of deterministic one-counter automata. We close this exponential complexity gap in Chapter 4 and provide an NL-completeness result for this problem.

3.3 Higher-Order Pushdown Systems

Higher-order pushdown automata generalize usual pushdown automata by allowing the stack to contain other stacks instead of only single symbols. They were introduced by Maslov [134] and independently by Damm and Goerdt [54]. These devices are closely related to recursion schemes, which are essentially simply typed λY terms that generate a single infinite tree. In fact, higher-order pushdown automata generate the same class of trees as *safe* higher-order recursion schemes [115]. Enjoying decidable μ-calculus theories, the class of trees generated by recursion schemes shows a lot of promise as a model for verifying higher-order functional programs [116, 148]. In the last couple years a lot of research has been devoted to verification of recursion schemes, which are in fact are strictly more expressive than safe ones [154, 152]. *Collapsible pushdown automata* extend higher-order pushdown automata by allowing "links" to the stack and are equi-expressive to the simply typed λY terms with respect to the trees they generate [87]. In this thesis we only concern ourselves with the systems generated by (non-collapsible) higher-order pushdown automata.

Before we define them we need to inductively define the set of k-*stacks*, for each $k \geq 1$, over some finite stack alphabet Γ with $[,] \notin \Gamma$ and where $\bot \notin \Gamma$ is a special *bottom-of-stack symbol*:

- A 1-*stack* is an element of $\Gamma^* \bot$.
- A $(k+1)$-*stack* is a finite sequence $[\alpha_1][\alpha_2] \cdots [\alpha_n]$, where $n \geq 1$ and α_i is a k-stack for each $i \in [1, n]$.

Let us denote by $\mathsf{Stacks}_k(\Gamma)$ the set of all k-*stacks* over Γ. The *empty order* k-*stack* $\bot_k$ is inductively defined as $\bot_1 \stackrel{\text{def}}{=} \bot$ and $\bot_{k+1} \stackrel{\text{def}}{=} [\bot_k]$ for each $k \geq 1$.

Over each 1-stack α we define the (partial) operation swap_w for each $w \in \Gamma^* \cup \Gamma^* \bot$ as

$$\mathsf{swap}_w(\alpha) \stackrel{\text{def}}{=} \begin{cases} w a_2 \cdots a_n \bot & \text{if } w \in \Gamma^*, \alpha = a_1 \cdots a_n \bot \in \Gamma^n \bot, n \geq 1 \\ w & \text{if } w \in \Gamma^* \bot \text{ and } \alpha = \bot, \text{ and} \\ \text{undefined} & \text{otherwise} \end{cases}$$

and

$$\mathsf{top}_1(\alpha) \stackrel{\text{def}}{=} \begin{cases} a_1 & \text{if } \alpha = a_1 \cdots a_n \bot \in \Gamma^n \bot, n \geq 1 \text{ and} \\ \bot & \text{otherwise.} \end{cases}$$

Let us define the partial operation $\mathsf{pop}_1(\alpha) \stackrel{\text{def}}{=} \mathsf{swap}_\varepsilon(\alpha)$ and for each k-stack $\alpha = [\alpha_1][\alpha_2] \cdots [\alpha_n]$

with $k \geq 2$ let us define:

$$\begin{aligned}
\mathsf{swap}_w(\alpha) &\stackrel{\text{def}}{=} [\mathsf{swap}_w(\alpha_1)][\alpha_2]\cdots[\alpha_n] \\
\mathsf{push}_k(\alpha) &\stackrel{\text{def}}{=} [\alpha_1][\alpha_1][\alpha_2]\cdots[\alpha_n] \\
\mathsf{push}_\ell(\alpha) &\stackrel{\text{def}}{=} [\mathsf{push}_\ell(\alpha_1)][\alpha_2]\cdots[\alpha_n] \quad \text{for each } 2 \leq \ell < k \\
\mathsf{pop}_k(\alpha) &\stackrel{\text{def}}{=} \begin{cases} [\alpha_2]\cdots[\alpha_n] & \text{if } n \geq 2 \\ \text{undefined} & \text{otherwise} \end{cases} \\
\mathsf{pop}_\ell(\alpha) &\stackrel{\text{def}}{=} [\mathsf{pop}_\ell(\alpha_1)][\alpha_2]\cdots[\alpha_n] \quad \text{for each } 2 \leq \ell < k \\
\mathsf{top}_k(\alpha) &\stackrel{\text{def}}{=} \alpha_1 \\
\mathsf{top}_\ell(\alpha) &\stackrel{\text{def}}{=} \mathsf{top}_\ell(\alpha_1) \quad \text{for each } 1 \leq \ell < k
\end{aligned}$$

Let $\mathsf{Op}_k \stackrel{\text{def}}{=} \{\mathsf{swap}_w \mid w \in \Gamma^* \cup \Gamma^*\bot\} \cup \{\mathsf{pop}_\ell \mid \ell \in [1,k]\} \cup \{\mathsf{push}_\ell \mid \ell \in [2,k]\}$ denote the set of *k-operations*. Note $\alpha \in \mathsf{Stacks}_k(\Gamma)$ and $\mathsf{op} \in \mathsf{Op}_k$ implies $\mathsf{op}(\alpha) \in \mathsf{Stacks}_k(\Gamma)$ if $\mathsf{op}(\alpha)$ is defined.

For each $k \geq 1$, an *order-k pushdown system (k-PDS)* is given by a tuple $\mathcal{P} = (Q, \mathsf{A}, \Gamma, \Delta)$, where

- Q is a finite set of *control states*,
- $\mathsf{A} \subseteq \mathsf{Act}$ is a finite set of *atomic actions*,
- Γ is a finite *stack alphabet*, and where
- $\Delta \subseteq Q \times (\Gamma \cup \{\bot\}) \times \mathsf{A} \times Q \times \mathsf{Op}_k$ is a finite set of *stack rewrite rules*, where each $(q, x, a, q, \mathsf{op}) \in \Delta$ satisfies
  - (i) $x = \bot$ and $\mathsf{op} = \mathsf{swap}_w$ implies $w \in \Gamma^*\bot$ and
  - (ii) $x \in \Gamma$ and $\mathsf{op} = \mathsf{swap}_w$ implies $w \in \Gamma^*$.

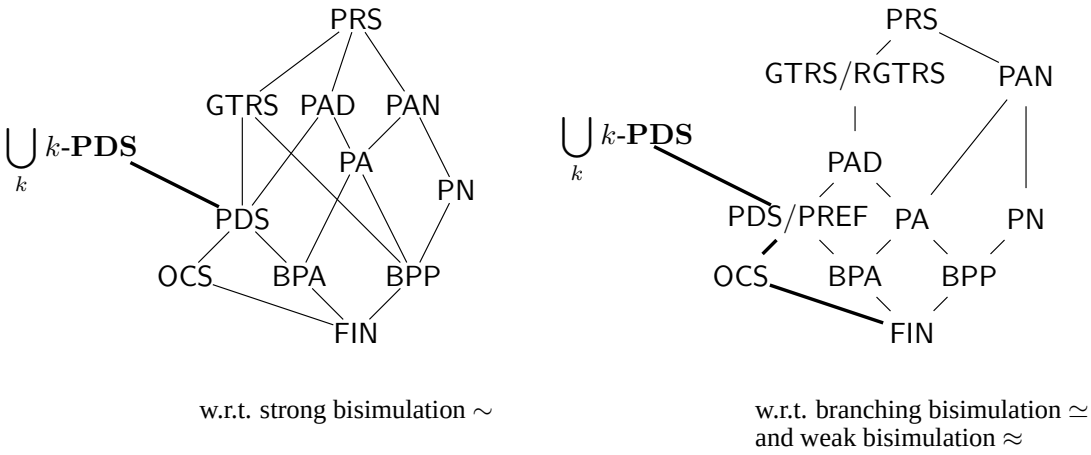
We abbreviate $(q, x, a, q', \mathsf{op}) \in \Delta$ by $qx \overset{a}{\hookrightarrow}_{\mathcal{P}} q'\mathsf{op}$.

The transition system of $\mathcal{P}$ is $\mathcal{T}(\mathcal{P}) \stackrel{\text{def}}{=} (Q \times \mathsf{Stacks}_k(\Gamma), \mathsf{A}, \{\overset{a}{\longrightarrow} \mid a \in \mathsf{A}\})$, where $q(\alpha) \overset{a}{\longrightarrow} q'(\alpha')$ if there is $qx \overset{a}{\hookrightarrow}_{\mathcal{P}} q'\mathsf{op}$ in Δ such that $\mathsf{top}_1(\alpha) = x$ and $\alpha' = \mathsf{op}(\alpha)$ for each $q, q' \in Q$, each $a \in \mathsf{A}$ and each $\alpha, \alpha' \in \mathsf{Stacks}_k(\Gamma)$.

Thus, states of $\mathcal{T}(\mathcal{P})$ are elements of $Q \times \mathsf{Stacks}_k(\Gamma)$ that we also denote as *configurations of $\mathcal{P}$*. We call a configuration $q_0(\alpha_0)$ of $\mathcal{P}$ *normed* if there exists some control state $q_f \in Q$ with $q_f(\bot_k) \overset{a}{\not\longrightarrow}$ (emits no a-transition) for each $a \in \mathsf{A}$, and such that every configuration $q(\alpha)$ with $q_0(\alpha_0) \longrightarrow^* q(\alpha)$ we have $q(\alpha) \longrightarrow^* q_f(\bot_k)$.

We refer to Figure 3.4 for integrating higher-order pushdown systems into Mayr's PRS hierarchy.

Model Checking. Reachability on order-k pushdown systems is complete for $(k-1)$-EXP [28]. It is worth mentioning that the transition graphs of higher-order pushdown systems have finite



w.r.t. strong bisimulation $\sim$

w.r.t. branching bisimulation $\simeq$ and weak bisimulation $\approx$

Figure 3.4: Refinement of Mayr's PRS hierarchy with GTRS, OCS and $\bigcup_k k$-PDS as Hasse diagrams (the top being the most expressive).

out-degree and decidable monadic second-order theories [32, 36]. We refer to [89, 34, 88] for various results on reachability and model checking LTL, CTL, EF and the modal μ-calculus on higher-order pushdown systems, lying between $(k-1)$-EXP and k-EXP.

Equivalence Checking. Undecidability of trace equivalence of higher-order pushdown systems is inherited from pushdown systems. Deciding equivalence of deterministic order-k pushdown automata is an interesting open problem, although some progress has been made on this by Stirling [179] on second-order simple grammars. We discuss undecidability of bisimulation equivalence of order-two pushdown systems in Section 8.1.

4 Equivalence checking of one-counter systems

In this chapter we determine the exact computational complexity of checking bisimilarity of one-counter systems and deterministic one-counter systems. Our main results in this chapter are that bisimilarity of (real-time) one-counter systems is PSPACE-complete and of deterministic one-counter systems is NL-complete both for real-time and even in the presence of ε-transitions. We also prove exact complexity bounds for regularity of general and deterministic one-counter systems, i.e. the question whether there exists a finite system that is bisimilar to the input system: we show that for general one-counter systems the problem is P-complete, whereas it is NL-complete for the deterministic case.

Mayr proved that weak bisimilarity of one-counter systems is undecidable [142]. Bisimilarity of one-counter systems has been proven decidable by Jančar [100]; the algorithm from [100] was analyzed to run in triply exponential space by Yen [200]. A PSPACE lower bound for this problem has been proven by Srba [175]. Our NL complexity results on equivalence of deterministic one-counter systems improve the previously best-known superpolynomial time upper bound by Valiant and Paterson [186] from 1975 that holds even in the presence of ε-transitions. For deterministic one-counter systems, the presence of ε-transitions makes the problem more complicated and indeed the proof technique drastically deviates from the real-time case that we discuss here in more detail.

From Section 4.1 to Section 4.7 we concern ourselves with the real-time case: we discuss in detail a PSPACE upper bound for bisimilarity of one-counter systems and an NL upper bound for deterministic one-counter systems. Finally we discuss in Section 4.7. the overall proof strategy for NL-completeness of equivalence deterministic one-counter automata (with possible ε-transitions) and point out the additional intricacy in comparison to the real-time case.

Bibliographic notes. The results on real-time one-counter systems have been published in the conference papers [15] (CONCUR 2010) in joint work with Stanislav Böhm and Petr Jančar and [14] (MFCS 2011) in joint work with Stanislav Böhm (which been merged in a journal paper [16] (Journal of Computer and System Sciences, 2013) and the result on general deterministic one-counter systems have appeared in the conference paper [17] (STOC 2013) in joint work with Stanislav Böhm and Petr Jančar.

4.1 A few notations and the main results

Let us introduce a few notations. Let us fix a transition system $\mathcal{T} = (S, \mathsf{A}, \{\stackrel{a}{\longrightarrow} \mid a \in \mathsf{A}\})$. For a subset $U \subseteq S$, by writing $s \stackrel{w}{\longrightarrow} U$ we mean that $s \stackrel{w}{\longrightarrow} u$ for some $u \in U$; similarly $s \longrightarrow^* U$ means that $s \longrightarrow^* u$ for some $u \in U$.

A transition system $\mathcal{T} = (S, \mathsf{A}, \{\stackrel{a}{\longrightarrow} | \ a \in \mathsf{A}\})$ is *image-finite* if for each $s \in S$ and each $a \in \mathsf{A}$ there are only finitely many $t \in S$ such that $s \stackrel{a}{\longrightarrow} t$; $\mathcal{T}$ is a *deterministic* transition system if for each pair $s \in S$, $a \in \Sigma$ there is at most one t such that $s \stackrel{a}{\longrightarrow} t$. We say $\mathcal{T} = (S, \mathsf{A}, \{\stackrel{a}{\longrightarrow} | \ a \in \mathsf{A}\})$ is a *finite transition system* if S is finite.

We recall that a union of bisimulations is again a bisimulation; $\sim$ is the greatest bisimulation (the union of all bisimulations on S), and it is an equivalence relation.

We also note that for *deterministic* transition systems bisimulation equivalence coincides with the variant of language equivalence called *trace equivalence*: $s \sim t$ if, and only if, for all words $w \in \Sigma^*$ we have $s \stackrel{w}{\longrightarrow} \Leftrightarrow t \stackrel{w}{\longrightarrow}$ (i.e., s and t enable the same words, also called traces). Let us briefly define deterministic one-counter systems. A one-counter system $\mathcal{P} = (Q, \mathsf{A}, \delta_0, \delta_{>0})$ is *deterministic* if for each $q \in Q$ and each $a \in \mathsf{A}$ there is at most one pair $(j, q') \in \{0, +1\} \times Q$ such that $(q, a, j, q') \in \delta_0$ and at most one pair $(j, q') \in \{-1, 0, +1\} \times Q$ such that $(q, a, j, q') \in \delta_{>0}$. Figure 4.1 shows an example of a transition system that is generated by a one-counter system.

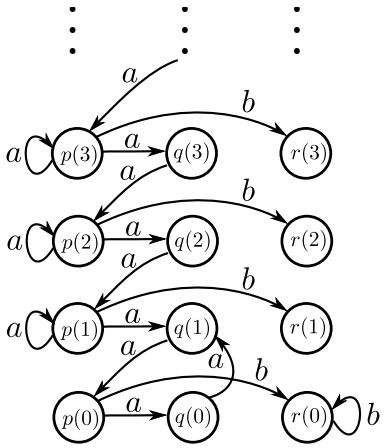


Figure 4.1: The transition system $\mathcal{T}(\mathcal{P})$ generated by some one-counter system $\mathcal{P}$

We also define the equivalences $\sim_0 \supseteq \sim_1 \supseteq \sim_2 \supseteq \cdots$ by the following inductive definition. We put $\sim_0 = S \times S$. For $k \geq 1$, we define $\sim_k \subseteq S \times S$ as

$$\sim_k \ \stackrel{\text{def}}{=} \ \{(s, s') \mid \forall s \stackrel{a}{\longrightarrow} t \exists s' \stackrel{a}{\longrightarrow} t' : t \sim_{k-1} t \text{ and } \forall s' \stackrel{a}{\longrightarrow} t' \exists s \stackrel{a}{\longrightarrow} t : t \sim_{k-1} t'\}$$

Note that $s \not\sim_1 t$ if, and only if, s, t enable different sets of actions. We recall the following standard facts (see, e.g., [143]).

Proposition 4.1 *For any image-finite transition system $\mathcal{T} = (S, \mathsf{A}, \{\stackrel{a}{\longrightarrow} | \ a \in \mathsf{A}\})$ we have:*

1. $\bigcap_{i \in \mathbb{N}} \sim_i \ = \ \sim$ *(hence $s \sim t$ if, and only if, $\forall i \in \mathbb{N} : s \sim_i t$).*
2. *If $|S| = k \in \mathbb{N}$ then $\sim_{k-1} = \sim_k = \sim$.* □

A crucial fact for Point 1. is that $\bigcap_{i\in\mathbb{N}} \sim_i$ is a bisimulation in image-finite systems. Point 2. is established by a standard partition refinement: when constructing $\sim_0$, $\sim_1$, ..., we reach a fixed point within k iterations.

The main problems we are interested in are the following ones.

BISI-OCS

INPUT: A one-counter system $\mathcal{P}$ and two configurations $p(m)$ and $q(n)$ of $\mathcal{P}$.
QUESTION: Does $p(m) \sim q(n)$ hold in $\mathcal{T}(\mathcal{P})$?

If the input is restricted to deterministic one-counter systems, then we get the problem BISI-DET-OCS.

BISI-DET-OCS

INPUT: A deterministic one-counter system $\mathcal{P}$ and two configurations $p(m)$ and $q(n)$ of $\mathcal{P}$.
QUESTION: Does $p(m) \sim q(n)$ hold in $\mathcal{T}(\mathcal{P})$?

We recall that the problem BISI-DET-OCS is, in fact, logspace equivalent to the classical language equivalence for deterministic (real-time) one-counter automata (with acceptance via final state for instance).

We will derive the following complexity results, assuming a standard input encoding, where the *counter values* m, n are given in binary. In fact, the lower bounds are not based on using large m, n: The lower NL bound for equivalence of determininistic one-counter systems trivially follows from the folklore digraph reachability problem and the PSPACE lower bound for the nondeterministic case was proven by Srba [175].

Hence all the results hold both for binary and unary encodings of the input configurations.

Theorem 4.2 ([16]) *The problem* BISI-OCS *is* PSPACE*-complete.* □

Theorem 4.3 ([16]) *The problem* BISI-DET-OCS *is* NL*-complete. Moreover, given a deterministic one-counter system $\mathcal{P}$ with k control states, if $p(0) \not\sim q(0)$ then $p(0) \not\sim_\ell q(0)$ where $\ell \leq pol(k)$ for a polynomial pol (that is independent of $\mathcal{P}$).* □

Our proof implicitly delivers to the following structural result.

Theorem 4.4 ([16]) *Given a one-counter system $\mathcal{P} = (Q, \mathsf{A}, \delta_0, \delta_{>0})$, the relation $\sim$ on $\mathcal{T}(\mathcal{P})$, i.e. the set $\{(p(m), q(n) \mid p(m) \sim q(n)\}$ is effectively semilinear, with the description size exponential in the size of $\mathcal{P}$.* □

By $\sim$ being semilinear we mean that the set $\{(m, n) \mid p(m) \sim q(n)\}$ is the union of finitely many linear subsets of $\mathbb{N} \times \mathbb{N}$ for each pair p, q. Recall that a subset $A \subseteq \mathbb{N}^m$ is linear if there is a base vector $b \in \mathbb{N}^m$ and periods $p_1, p_2, \ldots, p_\ell \in \mathbb{N}^m$ such that $A = \{b + c_1p_1 + c_2p_2 + \cdots + c_\ell p_\ell \mid c_1, c_2, \ldots, c_\ell \in \mathbb{N}\}$. Another view is that $\sim$ can be described by a formula in Presburger arithmetic.

We also consider the following regularity problems.

REG-BISI-OCS

INPUT: A one-counter system $\mathcal{P}$ and a configuration $p(m)$ of $\mathcal{P}$.
QUESTION: Is there a state f in some finite transition system such that $p(m) \sim f$?

REG-DET-OCS

INPUT: A deterministic one-counter system $\mathcal{P} = (Q, \mathsf{A}, \delta_0, \delta_{>0})$, and a configuration $p(m)$ of $\mathcal{P}$.
QUESTION: Is there a state f in a finite transition system such that $p(m) \sim f$?

Our results are the following.

Theorem 4.5 ([16]) REG-BISI-OCS *is* P*-complete.* □

Theorem 4.6 ([16]) REG-BISI-DET-OCS *is* NL*-complete.* □

4.2 The underlying finite system and consistent colorings

In this section we recall the ingredients of the proofs which already appeared in [100]. If not said otherwise, we (implicitly) refer to a fixed OCS $\mathcal{P} = (Q, \mathsf{A}, \delta_0, \delta_{>0})$, where $|Q| = k$. We first introduce the finite transition system $\mathcal{F}_\mathcal{P}$ which underlies $\mathcal{P}$, and the reachability distance of configurations of $\mathcal{P}$ to a (small finite) subset INC of configurations which are incompatible with $\mathcal{F}_\mathcal{P}$. The equality of the distances of $p(m), q(n)$ to INC is a necessary condition for $p(m) \sim q(n)$. Then we recall a natural correspondence between (bisimulation) relations on $Q \times \mathbb{N}$ and black-white colorings of the 3-dimensional space $\mathbb{N} \times \mathbb{N} \times (Q \times Q)$.

The underlying finite-state system $\mathcal{F}_\mathcal{P}$ and the set INC

Definition 4.7 *Given a one-counter system $\mathcal{P} = (Q, \mathsf{A}, \delta_0, \delta_{>0})$, the* underlying finite transition system $\mathcal{F}_\mathcal{P}$ *is $(Q, \mathsf{A}, \{\xrightarrow{a} \mid a \in \mathsf{A}\})$, where $q \xrightarrow{a} q'$ if, and only if, there is some j such that $(q, a, j, q') \in \delta_{>0}$.* □

Intuitively speaking $\mathcal{F}_\mathcal{P}$ behaves like $\mathcal{P}$ when the counter values are very large.

By just writing p (without any counter value) in such contexts we refer to the state p of the finite transition system $\mathcal{F}_\mathcal{P}$. We easily observe that $p(m) \sim_m p$. Recalling Proposition 4.1(2), we note that for $k = |Q|$ we have $p \sim q$ if, and only if, $p \sim_k q$ if, and only if, $p \sim_{k-1} q$. (Thus $\sim_{k-1}$ coincides with $\sim$ in $\mathcal{F}_\mathcal{P}$.)

We now define the set INC as the set of configurations of $\mathcal{P}$ which are "INCompatible" with $\mathcal{F}_\mathcal{P}$: when recalling that $k = |Q|$ we view $p(m)$ as incompatible with $\mathcal{F}_\mathcal{P}$ if there is no $q \in Q$ such that $p(m) \sim_k q$. The value $\mathsf{dist}(p(m))$ is the length of a shortest path in the transition system $\mathcal{T}(\mathcal{P})$ starting in $p(m)$ and ending in some $p'(m') \in$ INC.

Definition 4.8 ([16, 100]) *Assuming a one-counter system $\mathcal{P} = (Q, \mathsf{A}, \delta_0, \delta_{>0})$, where $|Q| = k$, we define* INC $\subseteq Q \times \mathbb{N}$ *and* $\mathsf{dist} : Q \times \mathbb{N} \to \mathbb{N} \cup \{\omega\}$ *as follows:*

- $\mathsf{INC} \stackrel{def}{=} \{p(m) \in Q \times \mathbb{N} \mid \forall q \in Q : p(m) \not\sim_k q\}$,
- $\mathsf{dist}(p(m)) \stackrel{def}{=} \inf\{\, \ell \mid \exists w \in \mathsf{A}^* : |w| = \ell \wedge p(m) \stackrel{w}{\longrightarrow} \mathsf{INC}\,\}$, *where* $\inf \emptyset = \omega$. □

The following proposition is not difficult to prove.

Proposition 4.9 ([16]) *The following holds:*

1. *If* $p(m) \in \mathsf{INC}$ *then* $m < k$.
2. *The membership in* INC *(given* $\mathcal{P}$ *and* $p(m)$*, is* $p(m) \in \mathsf{INC}$ *?) is* P*-complete; it is* NL*-complete when* $\mathcal{P}$ *is a deterministic one-counter system.* □

Figure 4.2 sketches a (shortest) path u from $p(m)$ to INC; we note that if, e.g., $n = k + |u|$ then $\mathsf{dist}(p(m)) < \mathsf{dist}(q(n))$, and thus $p(m) \not\sim q(n)$ as the next lemma states; the lemma also shows that $p(m) \sim_k q(n)$ implies $p(m) \sim q(n)$ when INC is unreachable from both $p(m)$ and $q(n)$.

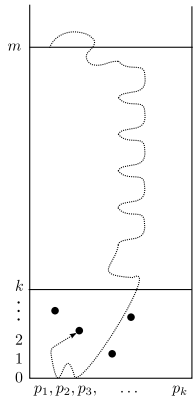


Figure 4.2: A path from $p(m)$ to INC

Lemma 4.10 ([16, 100])

1. *If* $\mathsf{dist}(p(m)) = \ell < \mathsf{dist}(q(n))$ *for some* $\ell \in \mathbb{N}$*, then* $p(m) \not\sim_{\ell+k} q(n)$*; hence* $p(m) \sim q(n)$ *implies* $\mathsf{dist}(p(m)) = \mathsf{dist}(q(n))$.
2. *If* $p(m) \not\longrightarrow^* \mathsf{INC}$, $q(n) \not\longrightarrow^* \mathsf{INC}$ *(so* $\mathsf{dist}(p(m)) = \mathsf{dist}(q(n)) = \omega$*) then* $p(m) \sim q(n)$ *if, and only if,* $p(m) \sim_k q(n)$. □

PROOF (SKETCH) Point 1. is immediate since Attacker can force the game to INC from $p(m)$ which then results in a pair of configurations that can surely not be bisimilar.

For Point 2. the "only-if"-direction is trivial. For the "if"-direction one can prove that due to $\sim_{k-1} = \sim_k$ in $\mathcal{F}_\mathcal{P}$ we have that the relation

$$R \quad = \quad \{\, (q_1(n_1), q_2(n_2)) \mid q_1(n_1) \sim_k q_2(n_2) \text{ and } q_1(n_1) \not\longrightarrow^* \mathsf{INC}, q_2(n_2) \not\longrightarrow^* \mathsf{INC}\,\}$$

is a bisimulation. ■

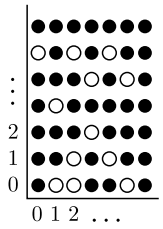


Figure 4.3: A colored plane for a pair (p, q)

Bisimulations and consistent colorings

We still refer to a fixed one-counter system $\mathcal{P} = (Q, \mathsf{A}, \delta_0, \delta_{>0})$, where $|Q| = k$. Recall that a bisimulation on $\mathcal{T}(\mathcal{P})$ is a relation R on $Q \times \mathbb{N}$ that satisfies certain closure properties. The relation R is thus a subset of $(Q \times \mathbb{N}) \times (Q \times \mathbb{N})$. We could depict such R in Figure 4.1, e.g. by adding special dotted arcs between $p(m), q(n)$ for $(p(m), q(n)) \in R$. But we use another geometrical presentation of a relation R on $Q \times \mathbb{N}$. E.g., Figure 4.3 can be viewed as presenting a (part of a) black-white coloring of points in the grid $\mathbb{N} \times \mathbb{N}$, corresponding to a fixed pair $(p, q) \in Q \times Q$: $(p(m), q(n)) \in R$ precisely for those (m, n) which are colored black. Putting together k^2 such colored 2-dimensional nonnegative-integer grids, one for each $(p, q) \in Q \times Q$, we get a coloring of the 3-dimensional grid, where the 3rd axis has only k^2 values. (We will later partition this 3-dimensional grid as in Figure 4.5.)

We now formalize the discussed notions.

Definition 4.11 ([16]) *For a (general) binary relation R on $Q \times \mathbb{N}$, by the* coloring χ_R *we mean the function $\chi_R : \mathbb{N} \times \mathbb{N} \times (Q \times Q) \to \{\bullet, \circ\}$ where $\chi_R(m, n, (p, q)) = \bullet$ if, and only if, $(p(m), q(n)) \in R$.*

Given (a coloring) $\chi : \mathbb{N} \times \mathbb{N} \times (Q \times Q) \to \{\bullet, \circ\}$, by R_χ we denote the relation $R_\chi = \{(p(m), q(n)) \mid \chi(m, n, (p, q)) = \bullet\}$. □

We note the one-to-one correspondence: $R_{\chi_R} = R$ and $\chi_{R_\chi} = \chi$.
We now introduce (local) consistency of colorings (given our fixed one-counter system), and we easily note that bisimulation relations correspond to consistent colorings. Roughly speaking, coloring $(m, n, (p, q))$ black is (locally) consistent if $(m, n, (p, q))$ (i.e., the pair $(p(m), q(n))$) is covered by the neighboring points (i.e. pairs) which are colored black.

Definition 4.12 ([16]) *A* coloring $\chi : \mathbb{N} \times \mathbb{N} \times (Q \times Q) \to \{\bullet, \circ\}$ *is* consistent in a point $(m, n, (p, q))$ *if either $\chi(m, n, (p, q)) = \circ$, or $\chi(m, n, (p, q)) = \bullet$ and the following (bisimulation) conditions are satisfied:*

(1) if $p(m) \xrightarrow{a} p'(m + j)$ (recall that $j \in \{-1, 0, 1\}$) then there is $q'(n + j')$ such that $q(n) \xrightarrow{a} q'(n + j')$ and $\chi(m + j, n + j', (p', q')) = \bullet$;

(2) if $q(n) \xrightarrow{a} q'(n + j')$ then there is $p'(m + j)$ such that $p(m) \xrightarrow{a} p'(m + j)$ and $\chi(m + j, n + j', (p', q')) = \bullet$.

Thus, consistency of χ in $(m, n, (p, q))$ is determined by the values of χ on

$$\mathsf{Neighbours}(m, n, (p, q)) \stackrel{def}{=} \{(m', n', (p', q')) \mid |m' - m| \leq 1, |n' - n| \leq 1\}.$$

A coloring χ *is* consistent *if it is consistent in each point* $(m, n, (p, q))$. □

The following proposition is obvious.

Proposition 4.13 ([16])

1. *$R \subseteq (Q \times \mathbb{N}) \times (Q \times \mathbb{N})$ is a bisimulation if, and only if, χ_R is consistent. (Hence χ is consistent if, and only if, R_χ is a bisimulation.)*
2. *The coloring $\chi_\sim$ (i.e. χ_R where $R = \{\, (p(m), q(n)) \mid p(m) \sim q(n) \,\}$) is the "darkest" consistent coloring, i.e.: if χ is consistent and $\chi(m, n, (p, q)) = \bullet$ then $\chi_\sim(m, n, (p, q)) = \bullet$.* □

4.3 Normal forms of paths

In this section we note some "normal forms" of (shortest) paths in $\mathcal{T}(\mathcal{P})$, for a one-counter system $\mathcal{P} = (Q, \mathsf{A}, \delta_0, \delta_{>0})$ with $|Q| = k$ that we fix from now on.

We first capture the section content at an intuitive level. As suggested by Figure 4.2, a shortest path from $p(m)$ to INC can assumed to be in the following normal form: it starts with a short prefix, where "short" means "bounded by a polynomial in k," then repeats a simple counter-decreasing cycle (when m is large), and then finishes with a short suffix (which might reach zero several times). Figure 4.4 illustrates a more general case, when the counter value n in the target $q(n)$ can be also large (unlike the case of INC where $n < k$). This also entails that the set $\{m \mid p(m) \longrightarrow^* \mathsf{INC}\}$ is "periodic" (for each $p \in Q$). We now make these observations precise.

We start with introducing the restriction $\xrightarrow{w}_+$ of $\xrightarrow{w}$ which captures the *positive paths*, where the counter value never becomes zero. We define $\xrightarrow{w}_+$ inductively: if $m \geq 1$ then $p(m) \xrightarrow{\varepsilon}_+ p(m)$; if $m \geq 1$ and $p(m) \xrightarrow{a} q(n) \xrightarrow{u}_+ q'(n')$ (which entails $n, n' \geq 1$) then $p(m) \xrightarrow{au}_+ q'(n')$. By writing $p(m) \longrightarrow^*_+ q(n)$ we mean that $p(m) \xrightarrow{w}_+ q(n)$ for some $w \in \mathsf{A}^*$.

We say that $v \in \mathsf{A}^+$ is a *cycle* if there is a $q \in Q$ such that $q(m) \xrightarrow{v}_+ q(m + d)$ (for some $m \in \mathbb{N}$, $d \in \mathbb{Z}$); d is called the *effect* of the cycle v, called also the *drop* if $d < 0$ and the *increase* if $d > 0$. A cycle $v \in \mathsf{A}^+$ is a *simple cycle* if no proper subword of v is a cycle.

The following proposition can be shown by a straightforward, though a bit technical, proof using a notion of a most effective (dropping) simple cycle on a path $p(m) \xrightarrow{w}_+ q(n)$ where $m \geq n + k^2$ which has already been proven in [186]. An example is shown in Figure 4.4.

Proposition 4.14 (A consequence from [186]) *Given $p(m), q(n)$, if u is a shortest word such that $p(m) \xrightarrow{u} q(n)$ (not necessarily $p(m) \xrightarrow{u}_+ q(n)$) then there is w such that $|w| = |u|$, $p(m) \xrightarrow{w} q(n)$, and*

- *$w = w_1(v_1)^{r_1} w_2 (v_2)^{r_2} w_3$, where*

- $|w_1| \leq k^3$, $|w_2| \leq k^3$, $|w_3| \leq k^3$,
- v_1 *is a dropping cycle and* $|v_1| \leq k$, *and*
- v_2 *is an increasing cycle and* $|v_2| \leq k$. □

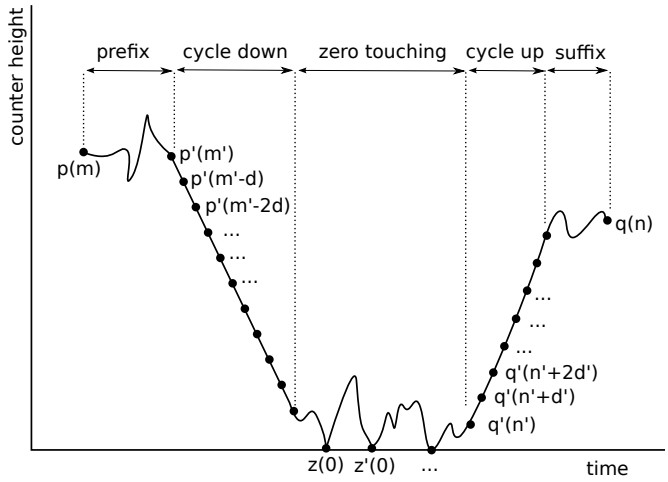


Figure 4.4: A shortest path from $p(m)$ to $q(n)$

We now fix a polynomial $\mathsf{poly}_0(k)$ (independent of the OCS $\mathcal{P}$) whose existence is captured by the following proposition. In fact, $\mathsf{poly}_0(k) \in O(k^3)$, but we do not perform a detailed analysis. Recall again Figure 4.2.

Proposition 4.15 ([16]) *There is some polynomial* $\mathsf{poly}_0 : \mathbb{N} \to \mathbb{N}$ *such that the following holds. If* $p(m) \longrightarrow^* \mathsf{INC}$ *then* $p(m) \xrightarrow{u} q(n)$ *for some* $q(n) \in \mathsf{INC}$, $u = u_1(u_2)^r u_3$, *and* $r \geq 0$, *where*

(1) $|u| = \mathsf{dist}(p(m))$,

(2) $|u_1 u_3| \leq \mathsf{poly}_0(k)$, *and*

(3) $|u_2| \leq k$ *and either* $u_2 = \varepsilon$ *or* u_2 *is a dropping cycle with the drop* $d \in [-k, -1]$.

One can thus write $\mathsf{dist}(p(m))$ *as a linear function in* m *where the coefficients and offsets are polynomially bounded in the size of the one-counter system: more precisely have*

$$\mathsf{dist}(p(m)) = c_1 + d_1 \frac{m - c_2}{d}$$

where $d_1 = |u_2| \leq k$, $1 \leq d \leq k$, $0 \leq c_1 \leq \mathsf{poly}_0(k)$, $-\mathsf{poly}_0(k) \leq c_2 \leq \mathsf{poly}_0(k)$. □

Now we make precise a (sufficiently small) periodicity of $\{m \mid p(m) \longrightarrow^* \mathsf{INC}\}$.

Lemma 4.16 ([16]) *Let us put* $\Delta = k!$. *For each configuration* $p(m)$ *with* $m > k + \mathsf{poly}_0(k)$ *we have* $p(m) \longrightarrow^* \mathsf{INC}$ *if, and only if,* $p(m + \Delta) \longrightarrow^* \mathsf{INC}$. □

4.4 Initial space, belts, and periodic background

We now explore $\chi_\sim$ for a given one-counter system $\mathcal{P} = (Q, \mathsf{A}, \delta_0, \delta_{>0})$ with $|Q| = k$ (recall Subsection 4.2); this gives rise to linear belts, as already noted and used in [109, 106, 100, 15].

By Lemma 4.10(1), $p(m) \sim q(n)$ implies $\mathsf{dist}(p(m)) = \mathsf{dist}(q(n))$. In case $\mathsf{dist}(p(m)) = \mathsf{dist}(q(n)) < \mathsf{poly}_0$, where poly_0 is from Prop. 4.15, this equality is only possible for polynomially many such $p(m), q(n)$.

For the infinitely many remaining pairs of configurations $p(m), q(n)$ the equality $\mathsf{dist}(p(m)) = \mathsf{dist}(q(n)) < \omega$ thus yields a "linear-belt constraint"

$$n \approx \frac{\alpha}{\beta} m \text{ where } \alpha, \beta \in [1, k^2], \tag{4.1}$$

with an error, called *offset*, $|n - \frac{\alpha}{\beta}m| \leq \mathsf{poly}_1(k)$ for a polynomial poly_1. We capture this in Lemma 4.18, after introducing the notion of belts (cf. Figure 4.5).

Definition 4.17 ([16]) *Assume integers $\alpha, \beta, h \geq 1$ such that α and β are relatively prime. The* belt *$\mathcal{B}(\alpha, \beta, h)$ is defined as $\mathcal{B}(\alpha, \beta, h) = \{(m, n) \in \mathbb{N} \times \mathbb{N} : |n - \frac{\alpha}{\beta}m| < \frac{h}{2}\}$. By the* slope *of the belt we mean the (rational) value $\frac{\alpha}{\beta}$; the value h is the (vertical)* thickness *of the belt.* □

The following lemma states that above some sufficiently large but polynomially bounded counter value each pair $(p(m), q(n))$ having the same finite distance to INC lies in precisely one of polynomially many belts, and moreover the different belts are so far away from each other that local consistency (with respect to any coloring) of any point in the one belt is not influenced by any point of the other belt.

Lemma 4.18 ([16]) *There are polynomials poly_1 and poly_2 (independent of the one-counter system $\mathcal{P}$) such that:*

(1) If $\max\{m, n\} > \mathsf{poly}_2(k)$ and $\mathsf{dist}(p(m)) = \mathsf{dist}(q(n)) < \omega$ then there are uniquely determined relatively prime $\alpha, \beta \in [1, k^2]$ for which $(m, n) \in \mathcal{B}(\alpha, \beta, \mathsf{poly}_1(k))$.

(2) If $\max\{m, n\} > \mathsf{poly}_2(k)$ and $(m, n) \in \mathcal{B}(\alpha, \beta, \mathsf{poly}_1(k))$ then for every pair $(m', n') \in \mathcal{B}(\alpha', \beta', \mathsf{poly}_1(k))$ for relatively prime $\alpha', \beta' \in [1, k^2]$ such that $\frac{\alpha'}{\beta'} \neq \frac{\alpha}{\beta}$ we have $|m - m'| > 1$ or $|n - n'| > 1$. □

The next definition partitions our 3-dimensional grid (cf Figure 4.5); here and further we assume that poly_0, poly_1, poly_2 are fixed polynomials guaranteed by Lemma 4.16 and 4.18, and also that $\mathsf{poly}_2(k) \geq k + \mathsf{poly}_0(k)$ (for all k).

Definition 4.19 ([16]) *We partition $\mathbb{N} \times \mathbb{N} \times (Q \times Q)$ into the following sets:*

- INITIALSPACE*: all $(m, n, (p, q))$ such that $m, n \leq \mathsf{poly}_2(k)$.*
- BELTSPACE*: all $(m, n, (p, q))$ outside the initial space such that $(m, n) \in \mathcal{B}(\alpha, \beta, \mathsf{poly}_1(k))$ for some (relatively prime) $\alpha, \beta \in [1, k^2]$.*
- BACKGROUND: *all remaining $(m, n, (p, q))$.* □

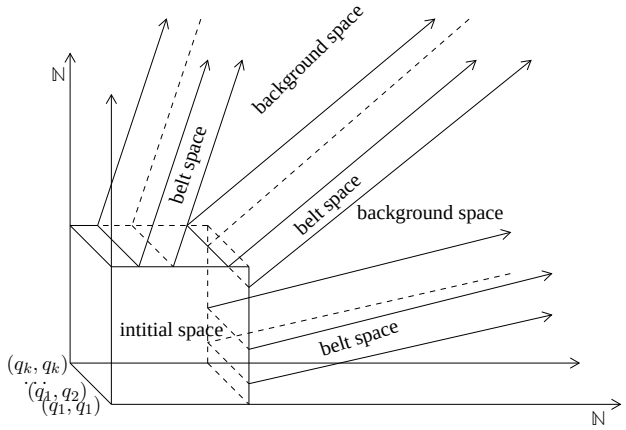


Figure 4.5: Partition of our 3-dimensional grid

We easily observe that we can compute the belt-points in the vertical cut at a given m (cf. Figure 4.6):

Proposition 4.20 ([16]) *There is a polynomial-time algorithm which, given a* OCS $\mathcal{P}$ *and* m*, computes all (polynomially many) points* $(m, n, (p, q))$ *in* INITIALSPACE *and* BELTSPACE. □

We note that Lemma 4.18 implies for each $(m, n, (p, q))$ in BACKGROUND that $p(m) \sim q(n)$ if, and only if, $p(m) \sim_k q(n)$ and $p(m) \not\longrightarrow^* \mathsf{INC}$, $q(n) \not\longrightarrow^* \mathsf{INC}$. Hence Lemma 4.16 then implies the following corollary.

Corollary 4.21 ([16]) *The coloring* $\chi_\sim$ *is periodic on* BACKGROUND*, with the period* $\Delta = k!$ *in both the horizontal and vertical directions, i.e.: if* $(m, n, (p, q))$ *and* $(m + i\Delta, n + j\Delta, (p, q))$ *belong to* BACKGROUND *then* $\chi_\sim(m, n, (p, q)) = \chi_\sim(m + i\Delta, n + j\Delta, (p, q))$. □

4.5 A polynomial space algorithm for bisimilarity of one-counter systems

Due to the general fact PSPACE = NPSPACE it suffices to show a *nondeterministic* polynomial space algorithm for BISI-OCS, as well as for some subproblems like the following one. Recall that INC can be constructed in polynomial time (Prop. 4.9).

Proposition 4.22 ([16]) *There is a polynomial time algorithm* ALG_1 *which, given a one-counter system* $\mathcal{P}$ *and a configuration* $p(m)$*, decides if* $p(m) \longrightarrow^* \mathsf{INC}$. □

In fact, later we strengthen the claim but the above form of Prop. 4.22 is sufficient for the moment.

Corollary 4.23 ([16]) *There is a polynomial space algorithm* ALG_2 *which, given a one-counter system* $\mathcal{P}$ *and* $(m, n, (p, q))$ *in* BACKGROUND*, decides if* $p(m) \sim q(n)$ *(i.e., if* $p(m) \sim_k q(n)$ *and* $p(m) \not\longrightarrow^* \mathsf{INC}$, $q(n) \not\longrightarrow^* \mathsf{INC}$*).* □

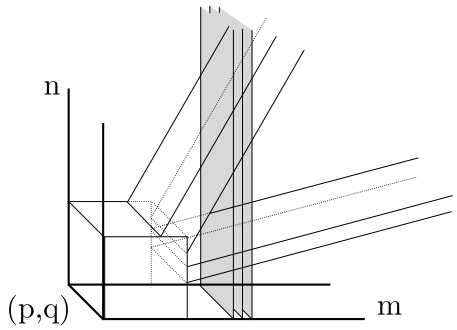


Figure 4.6: Vertical window of width 3

We now consider the following nondeterministic algorithm ALG for BISI-ROCA; it uses the algorithm ALG_2 from Corollary 4.23 as a subprocedure, and refers to the polynomials poly_1, poly_2 which were fixed before Definition 4.19. (Recall Figure 4.5; $\mathsf{poly}_1(k)$ is the vertical thickness of the belts and $\mathsf{poly}_2(k)$ determines the initial space.)

Before we give the algorithm for deciding BISI-OCS in polynomial space, let us explain it on an intuitive level. In case the initial pair of configurations lies in BACKGROUND, we call ALG_2 to get the definite answer. So let us hence assume the initial pair of configurations lies in INITIALSPACE $\cup$ BELTSPACE. The algorithm will guess a coloring of the 3D space that is locally consistent and is compatible with the determined colors from the background (that we obtain quickly from ALG_2).

Starting from $r = 0$ our algorithm will guess some subset of the "vertical window" of points $(m, n, (p, q))$ with $m = r$ that are in INITIALSPACE $\cup$ BELTSPACE. Since there are only polynomially many belts each of polynomial thickness and since the initial area is polynomially bounded, only a polynomially sized subset thus has to be guessed. The idea is that precisely the points in this subset will be the points in this small vertical window that are (guessed to be) colored with color $\bullet$, thus all other points in the same vertical window will be colored $\circ$.

Provided $r \geq 1$ our algorithm ALG will store in its memory only the guessed subsets of exactly three such consecutive windows, namely for $r - 1, r, r + 1$ as depicted in Figure 4.6. The algorithm then efficiently verifies on the fly that these guessed subsets are locally consistent possibly by looking at the background coloring dictated by ALG_2. If local consistency fails, then our guessing was wrong. Before r is then increased by one, the subset for window $r - 1$ can safely be deleted from the memory and the new subset of the vertical window $(r+1)+1 = r+2$ is guessed (but the subsets for windows r and $r + 1$ are kept). Since the background coloring is periodic and the thickness and number of belts is polynomially bounded, one can prove that if such a guessing of subsets of three vertical windows can be successfully carried through without ever violating local consistency *for exponentially* many consecutive r starting from $r = 0$ (this exponential bound will be denoted Δ' below), then this guessing can in fact be prolonged ad infinitum and thus we have guessed a coloring that is locally consistent. Of course, in our guessed coloring, we have to make sure that the initial pair of configurations is colored $\bullet$. Let us list algorithm ALG.

Algorithm ALG for deciding BISI-OCS

Input: A one-counter system $\mathcal{P} = (Q, \mathsf{A}, \delta_0, \delta_{>0})$, and two configurations $p_0(m_0), q_0(n_0)$.

1. If $(m_0, n_0, (p_0, q_0))$ is in BACKGROUND (recall Proposition 4.20) then use ALG_2 and otherwise go to the next step.

2. Compute the sufficiently large and exponentially bounded Δ' (we omit the exact value of Δ' here)
   and for $r = 0, 1, 2, \ldots, \max\{poly_2(k), m_0\} + \Delta'$ do the following (cf. Figure 4.6):
   - Using the algorithm from Proposition 4.20, nondeterministically choose a subset GUESS_r of the intersection of the vertical cut at point r with the belt and/or initial space; hence

     $$\text{GUESS}_r \subseteq \{(m, n, (p, q)) \mid m = r\} \cap \Big(\text{INITIALSPACE} \cup \text{BELTSPACE}\Big) ;$$

     its elements are deemed to be colored black while the other points in the initial and belt space of this vertical cut are white; the color of the background points is deemed to correspond to $\chi_\sim$. If $r = m_0$ then $(m_0, n_0, (p_0, q_0))$ must be in GUESS_r.
   - If $r \geq 1$ then check consistency of all points in GUESS_{r-1}, using the chosen GUESS_{r-2} (if $r \geq 2$), GUESS_{r-1}, GUESS_r, and the assumed $\chi_\sim$ on BACKGROUND (recall Definition 4.12). This task can require to call ALG_2 for finding the value of $\chi_\sim$ for the points in BACKGROUND which are neighbors of the initial and/or belt space. If a consistency test fails, this run of ALG fails.

3. (If $r = \max\{\mathsf{poly}_2(k), m_0\} + \Delta'$ is successfully processed then)
   halt with the answer $p_0(m_0) \sim q_0(n_0)$.

Lemma 4.24 ([16]) ALG *is a nondeterministic polynomial space algorithm deciding* BISI-OCS. □

4.6 A nondeterministic logspace algorithm for equivalence of deterministic one-counter systems

We now assume a fixed *deterministic* OCS $\mathcal{P} = (Q, \mathsf{A}, \delta_0, \delta_{>0})$, where $|Q| = k$, generating the deterministic transition system $\mathcal{T}(\mathcal{P}) = (Q \times \mathbb{N}, \mathsf{A}, \{\xrightarrow{a} \mid a \in \mathsf{A}\}$. We note that the transition system $\mathcal{T}(\mathcal{P}) \times \mathcal{T}(\mathcal{P})$, where $(p(m), q(n)) \xrightarrow{a} (p'(m'), q'(n'))$ if, and only if, $p(m) \xrightarrow{a} p'(m')$ and $q(n) \xrightarrow{a} q'(n')$, is also deterministic. We easily observe that $p(m) \not\sim q(n)$ if, and only if, there is some $w \in \Sigma^*$ such that $(p(m), q(n)) \xrightarrow{w} (p'(m'), q'(n'))$ where $p'(m') \not\sim_1 q'(n')$. Hence the question of equivalence in $\mathcal{T}(\mathcal{P})$ reduces to a reachability question in the deterministic transition system $\mathcal{T}(\mathcal{P}) \times \mathcal{T}(\mathcal{P})$. We write a path $(p(m), q(n)) \xrightarrow{w} (p'(m'), q'(n'))$ rather as $(m, n, (p, q)) \xrightarrow{w} (m', n', (p', q'))$, referring to our 3-dimensional space. Figure 4.7 sketches

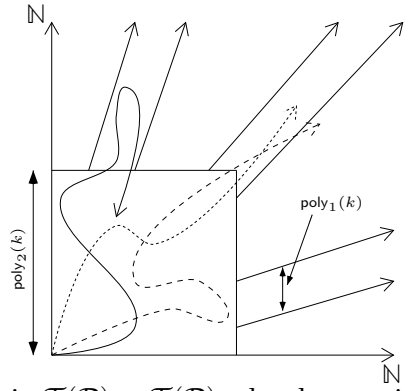


Figure 4.7: Examples of paths in $\mathcal{T}(\mathcal{P}) \times \mathcal{T}(\mathcal{P})$ related to a pair $p(0), q(0)$ and various words $w \in \mathsf{A}^*$.

some such paths when starting with $m = n = 0$. (The figure is simplified, it does not show the third dimension; hence the respective pairs of states are not depicted.)

A crucial fact for our result can be informally expressed as follows. When a (long) segment of such a path is inside some belt with slope $\frac{\alpha}{\beta}$ (where $\alpha, \beta \in [1, k^2]$) then the segment can be viewed as a computation-path of *one* deterministic one-counter system: we can imagine that the current point $(m, n, (p, q))$ is represented by m in the counter, while (p, q) and the (rational) offset $n - \frac{\alpha}{\beta}m$ (with the absolute value bounded by $\frac{1}{2}\mathsf{poly}_1(k)$) is stored in the finite control-unit. The number of possible offsets is polynomially bounded in k, and a long path that corresponds to w inside the belt can be replaced with a normal-form path that corresponds to $v_1(v_2)^r v_3$, by using (the "going up" form of) Proposition 4.14, where v_2 is a cycle for the deterministic OCS with the (polynomially) larger set of control states. In other words, any minimal distinguishing witness can visit each belt only for polynomially many steps.

Using the above fact, a straightforward analysis shows that the shortest words witnessing nonequivalence are polynomially bounded, in the case of nonequivalent pairs in INITIALSPACE. If we also allow large m, n (written in binary) in the input $p(m), q(n)$ then a shortest witness of the fact $p(m) \not\sim q(n)$ can be exponential (in the input size) but it is also at most exponential and its existence can be verified in nondeterministic logarithmic space, using the normal forms of paths and standard algorithms for arithmetic operations. These are the key ingredients for proving Theorem 4.3.

4.7 Regularity problems

We now prove Theorems 4.5 and 4.6. We assume a fixed OCS with k states. The next proposition is a variant of saying that $p(m)$ is nonregular if, and only if, the set $\{q(n) \mid p(m) \longrightarrow^* q(n) \longrightarrow^* \mathsf{INC}\}$ is infinite.

Proposition 4.25 ([16]) *$p(m)$ is not regular if, and only if, there is a $q \in Q$ such that $p(m) \longrightarrow^* q(m + 2k) \longrightarrow^* \mathsf{INC}$.* □

PROOF We recall that if $q(n) \not\longrightarrow^* \mathsf{INC}$ then $q(n) \sim r$ for some r in $\mathcal{F}_{\mathcal{P}}$. Hence "only if" is obvious.
For the "if" direction we note that a path $p(m) \xrightarrow{u_1} q(m+2k) \xrightarrow{u_2} \mathsf{INC}$ can be written $p(m) \xrightarrow{u_{11}} q_1(m+k) \xrightarrow{u_{12}} q(m+2k) \xrightarrow{u_{21}} q_2(m+k) \xrightarrow{u_{22}} \mathsf{INC}$ where all states=configurations in the segment $q_1(m+k) \xrightarrow{u_{12}} q(m+2k) \xrightarrow{u_{21}} q_2(m+k)$ have the counter values at least $m+k$. The first segment must contain an increasing cycle, the second a dropping cycle. By pumping the cycles appropriately we get infinitely many states reachable from $p(m)$ which have ever larger distance to INC. ■

It is not surprising that membership problem for INC is P-complete for (general) OCS, and NL-complete for deterministic OCS. Hence from Proposition 4.25 one can easily show that the problem REG-BISI-OCS is in P, and REG-BISI-DET-OCS in NL. Since REG-BISI-DET-OCS is obviously NL-hard (by a reduction from the reachability in directed finite graphs), with the next lemma one can finish the proofs of Theorems 4.5 and 4.6. In the lemma we only use OCS with *weak zero-tests* (like in Petri nets): we say that a OCS $\mathcal{P} = (Q, \mathsf{A}, \delta_0, \delta_{>0})$ is a *one-counter net* if for $x \in \{0,1\}$ we have $(q,a,x,q') \in \delta_0$ only if $(q,a,x,q') \in \delta_{>0}$.

Lemma 4.26 ([16]) REG-BISI-OCS *is* P*-hard, even when restricted to one-counter nets.* □

4.8 Equivalence of general deterministic one-counter automata

Let us discuss the crucial technical difficulties of proving an NL upper bound for the equivalence of deterministic one-counter systems/automata in which ε-transitions may occur (in a deterministic fashion, of course); we call them *doca* in the following.

Doca were first studied by Valiant and Paterson in 1975 [186]; they showed that equivalence is decidable in time $2^{O(\sqrt{n \log n})}$, and a simple analysis of their proof reveals that the equivalence problem is in PSPACE. The problem is easily shown to be NL-hard, there has been an exponential gap for this problem. There were attempts to settle the complexity of the doca equivalence problem but the problem proved to be intricate. Though doca are perhaps not a notorious computational device, their close relation to finite automata and deterministic pushdown automata (dpda) has motivated us to tackle this research problem. Establishing NL-completeness for the real-time case as discussed in Section 4.6 was a first step but it is far from clear if and how the proof can be extended to the general case.

One reason of the intricacy seems to be that a doca can exhibit a behavior with exponential periodicity, demonstrated by the following example (taken from [186]). We take a family $(\mathcal{P}_n)_{n \geq 1}$ of docas, where each doca $\mathcal{P}_n$ accepts the regular language $L_n = \{a^m b_i \mid 1 \leq i \leq n, m \equiv 0 \pmod{p_i}\}$, where p_i denotes the i^{th} prime number. The index of the Myhill-Nerode congruence of L_n is obviously $2^{\Omega(n)}$ but we can easily construct such $\mathcal{P}_n$ with $O(n^2 \log n)$ states. The example also demonstrates that doca are exponentially more succint than their real-time variant, since one can prove that real-time doca accepting L_n have $2^{\Omega(n)}$ states. Doca are also strictly more expressive than their real-time variant. Analogous expressiveness and succinctness results hold for dpda and real-time dpda, respectively.

When thinking of an underlying finite transition system of a general deterministic one-counter system one notices that it has exponentially many states, since the residue class modulo the *least common multiple* of all popping cycle counter effects has to be taken into account. In our recent work, we were able to cope with this problem.

In [17] we show that language equivalence of doca is NL-complete (while language inclusion is well-known to be undecidable); this closes the exponential complexity gap that has existed since the 1970s when doca were introduced. Our approach helps to answer related questions as well; e.g., regularity of the language accepted by a given doca can easily shown to be NL-complete.

We remark that in [12, 159] it is stated that equivalence of doca can be decided in polynomial time. Unfortunately, the proofs provided in [12, 159] were not exact enough to be verified, and they raise several questions which are unanswered to date.

Overview of the proof in [17]. Instead of defining doca clasically as restricted dpda, we use a convenient equi-succinct way where we partition the control states (and thus the configurations) into *stable states*, in which the automaton waits for a letter to be read, and into *reset states*, in which the counter is reset to zero; in the latter case the residue class of the current counter value modulo the number, called a *period*, specified by the current reset state determines the successor (stable) state. The periods correspond to the lengths of classical popping ε-cycles. We explore in [17] *trace equivalence*, i.e. the classical language equivalence where all states are viewed as accepting. We use a natural notion of the *equivalence level*, the *eqlevel* for short, of two configurations, corresponding to the length of a shortest non-equivalence witness word, and stipulated to be ω when the configurations are equivalent. For proving this result, we show that the eqlevels of two non-equivalent zero configurations are *small*, by which we mean that they are bounded by a polynomial (in the size of the given doca).

The only ingredient of our proof which we take directly from the previous works is a *cyclic form* of shortest positive paths in the transition system $\mathcal{T}(\mathcal{P})$ generated by a doca $\mathcal{P}$; this basic, but technical, fact was proven already in [186].

The central notion in our proof in [17] is the *extended deterministic transition system* $\mathcal{T}_{ext}(\mathcal{P})$ that is attached to a doca $\mathcal{P}$. Besides the standard transition system $\mathcal{T}(\mathcal{P})$, the extended system includes an underlying finite deterministic transition system that might be exponentially large in the size of $\mathcal{P}$ and that captures the *special mode* behavior of $\mathcal{P}$. The special mode (which can be seen as an abstraction of the behavior of the one-counter system by only remembering the residue class with respect to every occuring period of the reset states, but not the actual counter value itself) mimics the normal mode and is switched to the normal mode whenever a reset state is visited. The only difference is that when the zero counter value is reached (without a reset) then a multiple of all periods of the reset states is silently added to the counter; thus the counter never becomes zero in the special mode (until a reset state is visited and the normal mode applies). The above mentioned special finite system arises naturally once we note that the behavior of a special mode configuration depends on the residue classes of the counter value modulo the periods of the reset states, and not on the concrete counter value itself.

Each normal configuration $p(m)$ (where p is the control state and m is the counter value) thus has the special mode counterpart $\bar{p}(m)$. The *crucial novelty* of our approach consists in an explicit definition of the above $\mathcal{T}_{ext}(\mathcal{P})$ and in a detailed analysis of the *quadruple* (b, ℓ, r, o) associated with any pair of configurations $(p(m), q(n))$ as follows (here lev stands for *eqlevel*):

$b = \mathsf{lev}(p(m), q(n))$ (Basic), $\ell = \mathsf{lev}(p(m), \bar{p}(m))$ (Left), $r = \mathsf{lev}(q(n), \bar{q}(n))$ (Right), $o = \mathsf{lev}(\bar{p}(m), \bar{q}(n))$ (mOd). A simple fact that $\min\{b, \ell, r, o\}$ must be equal to at least two components of (b, ℓ, r, o) turns out to be very useful.

For each non-equivalent pair $(p_0(m_0), q_0(n_0))$ with a shortest non-equivalence witness word w we define $(p_i(m_i), q_i(n_i))$ as the (stable) pair such that $p_0(m_0)$ is transformed to $p_i(m_i)$, and $q_0(n_0)$ is transformed to $q_i(n_i)$ after having read the prefix of w of length i. Each $(p_i(m_i), q_i(n_i))$ has the associated quadruple (b_i, ℓ_i, r_i, o_i), and we note that $b_i = b_0 - i$. Though we have theoretically exponentially many pairs $(\bar{p}(m), \bar{q}(n))$, it is easy to show that the set of eqlevels $\{e \mid e = \mathsf{lev}(\bar{p}(m), \bar{q}(n))\}$ is small (i.e., its cardinality is bounded by a polynomial); in other words, there are only few possible values o_i. A straightforward analysis also shows that for each natural number g there are only few $p(m)$ such that $\mathsf{lev}(p(m), \bar{p}(m)) = g$. By using such observations we derive that if $m_0 = n_0 = 0$ then there are only few i such that $\ell_i \neq r_i$. Roughly speaking, $\ell_i = r_i < \omega$ implies that the counter values m_i and n_i are in one of only few linear relations. Hence if our sequence $(p_0(m_0), q_0(n_0))$, $(p_1(m_1), q_1(n_1))$, $(p_2(m_2), q_2(n_2))$, $\ldots$ (with $m_0 = n_0 = 0$) were long then it would have a long segment where $\ell_i = r_i$ and the values m_i, n_i are increasing on the whole. We contradict the existence of such a long segment by another use of cyclicity and the properties of the quadruples (b, ℓ, r, o).

A complete version of this work is available [16].

Theorem 4.27 ([17]) *Equivalence of deterministic one-counter automata is* NL*-complete.* □

5 Branching time model checking on one-counter systems and a new lower bound technique

In this section our starting point is to analyze the computational complexity of model checking CTL on one-counter systems and succinctly presented one-counter systems.

In Section 5.1 we provide a fixed one-counter system for which model checking CTL is PSPACE-hard. Complementing the latter lower bound, we discuss an upper bound result in Section 5.2: we provide a polynomial time algorithm for model checking fixed one-counter systems against input CTL formulas that satisfy a certain syntactical restriction – they have to have a fixed "leftward until depth", a notion to be made more precise below.

In Section 5.3 we discuss a new technique for proving lower bounds which it is inspired from two deep results from complexity theory. This technique is applied to prove that there is already a fixed CTL formula for which model checking one-counter systems is PSPACE-hard; a corresponding EXPSPACE-hardness result is proven for succinct one-counter systems. Moreover, one can prove that EF model checking of one-counter systems is hard for P^{NP}. We further apply the latter lower bound technique in Section 5.4 to prove that it is PSPACE-hard to decide whether one can reach a specific zero configuration in a one-counter Markov decision process with probability arbitrarily close to 1. In Section 5.5 we apply our lower bound technique to prove that (i) the data complexity of model checking CTL on 2-clock timed automata is PSPACE-hard and (ii) reachability of very basic 2-clock timed automata with modulo tests is PSPACE-hard.

Bibliographic notes. The results in this chapter have been published in [79] (STACS 2010) and will appear as a journal version in [80] (SIAM Journal of Computing) both in joint work with Markus Lohrey. The only exception is the EXPSPACE-hardness hardness result of model checking succinct one-counter systems against fixed CTL formulas which have been published in [73] (ICALP 2010) in joint work with Christoph Haase, Joël Ouaknine and James Worrell.

5.1 Hardness of Expression Complexity

The goal of this section is to prove that model checking CTL is PSPACE-hard already over a fixed OCS. The overall proof will be provided by a reduction from the well-known PSPACE-complete problem to decide validity of quantified boolean formulas (QBF). Instead of discussing all details here, we only pinpoint the decisive step in the lower bound proof. Most crucial is to be able to compute a family of CTL formulas $(\varphi_i)_{i\geq 1}$ such that over the fixed $\mathcal{P}_0$ that is depicted in Figure 5.1 we can express (non-)divisibility by 2^i. We implicitly assume that each control state

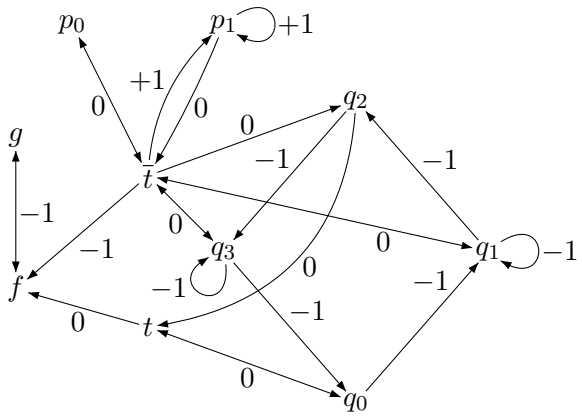


Figure 5.1: The one-counter system $\mathcal{P}_0$ for which CTL model checking is PSPACE-hard.

x (like $g, f, \ldots$) in $\mathcal{P}_0$ has a self-loop that is labelled by x and that does not change the current counter value (these loops are *not* depicted in Figure 5.1). Moreover the transitions that one can see in Figure Figure 5.1 are all labeled with some dummy symbol, say \$. In our formulas below, we will use EX as an abbreviation for $\langle\$\rangle$ and that x is an abbreviation for $\langle x\rangle\mathsf{true}$: in particular the formula x holds in $q(n)$ if and only if $q = x$.

We need the following simple fact which characterizes divisibility by powers of two.

Fact 5.1 ([80]) *Let $n \geq 0$ and $i \geq 1$. Then the following two statements are equivalent:*

- *2^i divides n.*
- *2^{i-1} divides n and $|\{1 \leq n' \leq n \mid 2^{i-1}$ divides $n'\}|$ is even.* □

The set of action labels of $\mathcal{P}_0$ in Figure 5.1 coincides with its control states plus the dummy symbol \$.

Note that both t and $\bar{t}$ are control states of $\mathcal{P}_0$. Now we define a family of CTL formulas $(\varphi_i)_{i\geq 1}$ such that for each $n \in \mathbb{N}$ we have that

- $(\mathcal{T}(\mathcal{P}_0), t(n)) \models \varphi_i$ if, and only if, 2^i divides n and
- $(\mathcal{T}(\mathcal{P}_0), \bar{t}(n) \models \varphi_i$ if, and only if, 2^i does *not* divide n.

On first sight, it might seem superfluous to let the control state t represent divisibility by powers of two and the control state $\bar{t}$ to represent non-divisibility by powers of two since CTL allows negation. By making use of Fact 5.1, we construct the formulas φ_i inductively. However the fact that we have *only one* family of formulas $(\varphi_i)_{i\geq 1}$ to express both divisibility and non-divisibility is a crucial technical subtlety that is necessary in order to avoid an exponential blowup in formula size: since the size of φ_i has to be polynomially bounded in i, we simply cannot afford φ_i to appear twice in an inductive definition of φ_{i+1}. First, let us define the auxiliary formulas $\text{test} = t \vee \bar{t}$ and $\varphi_\diamond = q_0 \vee q_1 \vee q_2 \vee q_3$. Think of $\varphi_\diamond$ to hold in those control states that altogether are situated in the "diamond" in Figure 5.1. We define

$$\varphi_1 \quad \stackrel{\text{def}}{=} \quad \text{test} \ \wedge \ \mathsf{EX}\,(f \wedge \mathsf{EF}(f \wedge \neg\mathsf{EX}g))\,.$$

Now assume $i > 1$. Then we define

$$\begin{aligned} \varphi_i &\stackrel{\text{def}}{=} \text{test} \wedge \mathsf{EX}\mu_i, \text{ where} \\ \mu_i &\stackrel{\text{def}}{=} \mathsf{E}(\varphi_\diamond \wedge \mathsf{EX}\varphi_{i-1})\mathsf{U}(q_0 \wedge \neg\mathsf{EX}q_1). \end{aligned}$$

Before we state that φ_i indeed expresses (non-)divisibility in Lemma 5.3, let us provide some informal explanation. Observe that φ_i can only be true either in control state t or $\bar{t}$. Note that in the formula μ_i the formula right to the until symbol expresses that we are in q_0 and that the current counter value is zero. Also note that the formula left to the until symbol requires that $\varphi_\diamond$ holds, i.e., we are always in one of the four "diamond control states". In other words, μ_i expresses that we decrement the counter by moving along the diamond control state (by possibly looping) and always check if $\mathsf{EX}\varphi_{i-1}$ holds (and hereby jumping back to t or $\bar{t}$ to check if φ_{i-1} holds), just until we are in q_0 and the counter value is zero. Since φ_{i-1} is only used once in φ_i, we get:

Fact 5.2 ([80]) $|\varphi_i| \in O(i)$. □

The following lemma states the correctness of the construction.

Lemma 5.3 ([80]) *Let $n \geq 0$ and $i \geq 1$. Then*

(1) $(\mathcal{T}(\mathcal{P}_0), t(n)) \models \varphi_i$ *if, and only if,* 2^i *divides* n.

(2) $(\mathcal{T}(\mathcal{P}_0), \bar{t}(n)) \models \varphi_i$ *if, and only if,* 2^i *does not divide* n. □

Building upon the above-mentioned gadget that allows us to test (non-)divisibility by powers of 2, we can provide (still with quite some technical effort) a polynomial time reduction from QBF to model checking the fixed $\mathcal{P}_0$ of Figure 5.1.

Theorem 5.4 ([80]) *For the fixed* OCS $\mathcal{P}_0$ *that is depicted in Figure 5.1 the following problem is* PSPACE*-hard.*
INPUT: A CTL *formula* φ.
QUESTION: Does $(\mathcal{T}(\mathcal{P}_0), f(0)) \models \varphi$ *hold?* □

5.2 Upper bounds for CTL model checking

When inspecting the PSPACE-hardness proof for the expression complexity of model checking CTL on fixed one-counter systems in the last section, we needed to construct formulas that had a non-constant nesting of the until operator. This section complements the latter result by providing a polynomial time algorithm for model checking fixed one-counter systems with respect to CTL formulas whose leftward until nesting depth is constantly bounded. For a CTL

formula φ we inductively define its *leftward until depth* $\mathsf{lud}(\varphi)$

$$\begin{aligned}
\mathsf{lud}(\mathtt{true}) &\stackrel{\text{def}}{=} 0\\
\mathsf{lud}(\neg\varphi) &\stackrel{\text{def}}{=} \mathsf{lud}(\varphi)\\
\mathsf{lud}(\varphi_1 \wedge \varphi_2) &\stackrel{\text{def}}{=} \max\{\mathsf{lud}(\varphi_1), \mathsf{lud}(\varphi_2)\}\\
\mathsf{lud}(\langle a\rangle\varphi) &\stackrel{\text{def}}{=} \mathsf{lud}(\varphi)\\
\mathsf{lud}(\mathsf{E}\varphi_1\mathsf{U}\varphi_2) &\stackrel{\text{def}}{=} \max\{\mathsf{lud}(\varphi_1)+1, \mathsf{lud}(\varphi_2)\}\\
\mathsf{lud}(\mathsf{E}\varphi_1\mathsf{WU}\varphi_2) &\stackrel{\text{def}}{=} \max\{\mathsf{lud}(\varphi_1)+1, \mathsf{lud}(\varphi_2)\}
\end{aligned}$$

Our main result states that model checking each fixed one-counter system against input CTL formulas of fixed leftward until depth is decidable in polynomial time. Its proof is based on a combinatorial analysis how for each control state q of a one-counter system $\mathcal{P}$ and for each CTL formula φ the set $\{n \in \mathbb{N} \mid (\mathcal{T}(\mathcal{P}), q(n)) \models \varphi\}$ behaves: one can prove that this set is ultimately periodic with an offset and period that is bounded polynomial in $|\varphi|$ and exponential only in $|\mathcal{P}| + \mathsf{lud}(\varphi)$.

Theorem 5.5 ([80]) *For every fixed one-counter system $\mathcal{P}$ and every $k \in \mathbb{N}$ the following problem is decidable in polynomial time:*
INPUT: A state $q(n)$ of $\mathcal{T}(\mathcal{P})$ and a CTL *formula φ with $\mathsf{lud}(\varphi) \leq k$.*
QUESTION: $(\mathcal{T}(\mathcal{P}), q(n)) \models \varphi$? □

Since every EF formula can be seen as a CTL of leftward until depth at most one, we obtain the following corollary.

Corollary 5.6 ([80]) EF *model checking for each fixed one-counter system is decidable in polynomial time.* □

5.3 A new lower bound technique

In this section we first concern ourselves with model checking *fixed* CTL formulas. For this we develop a lower bound technique that can also be used for other settings, like model checking or reachability of (extensions of) timed automata or deciding reachability objectives in one-counter Markov decision processes. Before we discuss the technique, we would briefly like to discuss why we are convinced that such a new technique is necessary, in particular for proving PSPACE-hardness for model checking fixed CTL formulas on one-counter systems. Let us describe the difficulties when aiming to prove PSPACE-hardness in two standard ways: either reducing from QBF or reducing from the membership problem of linearly space bounded Turing machines.

Firstly, when model checking a fixed CTL formula, one cannot expect a straightforward reduction from QBF in which the CTL formula depends on the QBF formula, simply because the number of nested negations of a fixed CTL formula is of course fixed again. The situation for model checking the modal μ-calculus is somewhat different since a single fixed point operator can indeed mimic alternation.

Secondly, let us discuss the hurdles when reducing from the word problem of a fixed linearly space bounded Turing machine $\mathcal{M}$ on some input $w \in \{0,1\}^n$. Essentially such a potential reduction has to provide a way to modify the current configuration (which is again a word from $\{0,1\}^n$, say). But how can we encode such a configuration on a counter? It is important to note that one can express already with a fixed EF formula some exponentially big periodic behavior on the counter. More precisely, there exists already a fixed EF formula φ such that for some control state q over some one-counter system $\mathcal{P}$ of size m we have that $\{n \in \mathbb{N} \mid (\mathcal{T}(\mathcal{P}), q(n)) \models \varphi\}$ is ultimately periodic with exponential periodicity (in m): simply introduce $O(\sqrt{m})$ cycles to $\mathcal{P}$ each of pairwise different prime number lengths and let the formula φ express that the current counter value is a multiple of all these prime numbers. This way of enforcing large values naturally leads us to the following definitions. Let p_i denote the i^{th} prime number. It is well-known that p_i is polynomially bounded in i; hence it requires only $O(\log i)$ bits for representing the i^{th} prime in binary.

For a number $0 \leq M < \prod_{i=1}^{m} p_i$ we define the *Chinese remainder representation* $\mathsf{CRR}_m(M)$ as the boolean tuple

$$\mathsf{CRR}_m(M) = (x_{i,r})_{i\in[1,m],0\leq r<p_i} \quad \text{with } x_{i,r} = \begin{cases} 1 & \text{if } M \bmod p_i = r \\ 0 & \text{otherwise.} \end{cases}$$

Usually the Chinese remainder representation of M is the tuple $(r_i)_{i\in[1,m]}$, where $r_i = M \bmod p_i$. Since the primes p_i will be always given in unary notation, there is no essential difference between this representation and our Chinese remainder representation.

In the spirit of proving lower bounds, the Chinese remainder representation of a number is a potential "data-structure" that allows us *to retrieve* information about an object out of exponentially many (depending on the input size n) via fixed CTL formulas. We emphasize the word *retrieve* here since the crucial point is that it is not all clear how to *modify* a number: Assume we encoded the current configuration of $\mathcal{M}$ in Chinese remainder representation and one would want to flip the i^{th} component of the boolean tuple, this would amount to *multiplying* the current counter value with a huge number. However, in a one-counter system one can only increment or decrement the current counter value or leave it as is. Analogously, if we encoded a configuration from $\{0,1\}^n$ classically, i.e. by the counter whose value would be between 0 and $2^n - 1$, and we would like to switch the i^{th} bit from 0 to 1, we would have to add 2^i to the counter, which one cannot achieve in one step with a one-counter system (only with a succinct one-counter system) and also not easily (if at all) in multiple steps without influencing the other bit positions.

5.3.1 Hardness of data complexity

The lower bound technique we develop is inspired by two deep results from complexity theory.

The **first result**, due to Chiu, Davida and Litow, states that one can transform a CRR-representation very efficiently into a binary representation: we denote by $\mathsf{BIN}_m(N)$ the m least significant bits in the binary representation of $N \in \mathbb{N}$.

Theorem 5.7 ([43]) *There is a logspace-uniform* NC^1*-circuit family* $(B_m((x_{i,r})_{i\in[1,m],0\leq r<p_i}))_{m\geq 1}$

such that for every $m \geq 1$, B_m has m output gates and

$$\forall\, 0 \leq M < \prod_{i=1}^{m} p_i : B_m(\mathsf{CRR}_m(M)) = \mathsf{BIN}_m(M \bmod 2^m).$$

By [91], we could replace logspace-uniform NC^1-circuits in Theorem 5.7 even by DLOGTIME-uniform TC^0-circuits. The existence of a P-uniform NC^1-circuit family for converting from CRR-representation to binary representation was already shown in [9].

The **second result** from complexity theory we use is the concept of *serializability* of complexity classes. Intuitively, a complexity class $\mathcal{C}_1$ is called $\mathcal{C}_2$-serializable (where $\mathcal{C}_2$ is another complexity class) if every language $L \in \mathcal{C}_1$ can be accepted in the following way: There exists a polynomial $p(n)$ and a $\mathcal{C}_2$-machine (or $\mathcal{C}_2$-circuit family) A such that $x \in L$ is checked in $2^{p(|x|)}$ many stages, which are indexed by the strings from $\{0,1\}^{p(|x|)}$. In stage $y \in \{0,1\}^{p(|x|)}$, A gets from the stage indexed by the lexicographic predecessor of y a constant number of bits $b_1, \ldots, b_c$ and computes from these bits, the index y and the original input x new bits $b'_1, \ldots, b'_c$ which are delivered to the lexicographic next stage. Cai and Furst proved that PSPACE is P-serializable [35]; in [90] Hertrampf, Lautemann, Schwentick, Vollmer and Wagner sharpened this to prove that AC^0-serializability is sufficient, cf. [193]. So let us state this theorem again.

Theorem 5.8 ([90]) PSPACE *is* AC^0*-serializable.* □

It is not stated in [90, 193] but not hard to prove that *logspace-uniform* AC^0 suffices for serializing PSPACE.

For our purpose, a slightly different definition of AC^0-serializability is useful: A language L is AC^0-serializable if there exists an nondeterministic finite automaton A over the alphabet $\{0,1\}$, a polynomial $p(n)$, and a logspace-uniform AC^0-circuit family $(C_n)_{n\geq 0}$, where C_n has exactly $n + p(n)$ many inputs and one output, such that for every $x \in \{0,1\}^n$ we have:

$$x \in L \iff \prod_{y \in \{0,1\}^{p(n)}} C_n(x,y) \ \in L(A), \tag{5.1}$$

where "$\prod$" is ordered with respect to the lexicographic order on $\{0,1\}^{p(n)}$ and for every $y \in \{0,1\}^{p(n)}$, $C_n(x,y)$ is either 0 or 1 (hence, $\prod_{y\in\{0,1\}^{p(n)}} C_n(x,y)$ is a binary string of length $2^{p(n)}$). We prove in [80] that this definition of AC^0-serializability is equivalent to the one in [90].

Combining the efficient translation from Chinese remainder representation to binary representation of a natural number (Theorem 5.7) allows us to restate a variant of serializability along Theorem 5.8 that is more tailored towards our purposes of proving PSPACE-hardness of model checking fixed CTL formulas over one-counter systems and related verification problems.

Theorem 5.9 ([80]) *For every language $L \subseteq \{0,1\}^*$ from* PSPACE *there exists a polynomial $p(n)$ and a nondeterministic finite automaton A over the alphabet $\{0,1\}$ such that the following holds: From a given input $x \in \{0,1\}^*$ with $|x| = n$ one can construct in logspace a boolean formula F with propositional variables $x_{i,r}$ ($i \in [p(n)]$ and $0 \leq r < p_i$) such that:*

$$x \in L \iff \prod_{M=0}^{2^m-1} F(\mathsf{CRR}_m(M)) \in L(A). \tag{5.2}$$

PROOF (SKETCH) The formula F essentially consists of the composition of the circuit C_n from (5.1) (which can be represented as a boolean formula) and an appropriate circuit B_m from Theorem 5.7 (which again can be represented as a boolean formula). ■

The proof of the following theorem, stating that the data complexity of model checking CTL on one-counter systems is PSPACE-hard, uses the characterization provided in (5.2).

Theorem 5.10 ([80]) *There exists a fixed* CTL *formula of the form* $\mathsf{EG}\psi$*, where* ψ *is an* EF *formula, such that the following problem is* PSPACE*-complete:*
INPUT: A one-counter system $\mathcal{P}$ *with control state* q*.*
QUESTION: Does $q(0) \models \mathsf{EG}\psi$ *hold in* $\mathcal{T}(\mathcal{P})$*?* □

PROOF (SKETCH) One can construct in polynomial time a one-counter system $\mathcal{P}$ that uses its counter to represent the M from (5.2). The states of the NFA A of $\mathcal{P}$ are represented in control states of $\mathcal{P}$. Moreover, when feeding the simulation of the NFA A with the current bit $F(\mathsf{CRR}_m(M))$, one needs to simulate the evaluation of the boolean formula F on $\mathsf{CRR}_m(M)$. The evaluation of $F(\mathsf{CRR}_m(M))$ can be done by adding to $\mathcal{P}$'s control states a possibility of traversing the syntax tree of F: The actual evaluation of F can be achieved by traversing this syntax tree which in turn can be accomplished by a fixed CTL formula – evaluating the atomic subformulas of F boils down to answering questions of the kind whether the current counter value is congruent r_i modulo p_i, which is easy to check by introducing a popping cycle of length p_i into the OCS. After the bit $F(\mathsf{CRR}_m(M))$ has been obtained, the NFA A is fed with this bit and M is incremented by one and the simulation continues until $M = 2^m - 1$ holds (this can be checked analogously). ■

By only making use of the efficient translation from Chinese remainder to binary representation (Theorem 5.7) we can prove that model checking EF on one-counter systems is hard for P^{NP}.

Theorem 5.11 *The following problem is* P^{NP}*-hard:*
INPUT: A one-counter system $\mathcal{P}$*, a control state* q *of* $\mathcal{P}$ *and an* EF *formula* φ*.*
QUESTION: Does $q(0) \models \varphi$ *hold in* $\mathcal{T}(\mathcal{P})$*?* □

PROOF (SKETCH) We reduce from the P^{NP}-complete problem MAX-LEX-SAT which asks whether the lexicographically maximal satisfying truth assignment of a given boolean formula $\alpha(x_1, \ldots, x_n)$ assigns the least significant variable x_n to 1. There is obviously a correspondence between the set of all truth assignments $\{0,1\}^{\{x_1,\ldots,x_n\}}$ and $\{0, \ldots, 2^n - 1\}$. In order to obtain the natural $j \in \{0, \ldots, 2^n - 1\}$ that corresponds to the lexicographically maximal truth assignment we jump (via the EF operator) from $q(0)$ to some configuration $q'(k)$ and test whether $k = j$ as follows: (i) test whether $k < 2^n$ by testing whether one cannot subtract from k some number such that the result is 2^n, (ii) test whether the truth assignment that corresponds to k satisfies α, (iii) test whether there does not exist a k' with $k < k' < 2^n$ whose truth assignment satisfies α in analogy to (ii), and finally (iv) test whether the least significant bit of k is 1. Points (i) to (iii) can be done by evaluating appropriate EF formulas that mimic a boolean formula that one obtains from the NC^1 circuit from Theorem 5.7 (the translation from Chinese remainder presentation to binary presentation), whereas Point (iv) is easy. ■

It is not clear whether P^{NP}-hardness already holds for a fixed EF formula. For fixed EF formulas we were only able to prove hardness for $\mathsf{P}_{\|}^{\mathsf{NP}}$ and containment in P^{NP} [82] (we discuss EF model checking on one-counter systems in Chapter 6).

In [73] we have studied the data complexity of model checking CTL on succinct one-counter systems and proved that it is EXPSPACE-complete. The proof of EXPSPACE-hardness is again inspired by serializability arguments stating that EXPSPACE is PSPACE-serializable (the concatentation in (5.1) now involves doubly exponentially many numbers). However, it is technically much more involved than the PSPACE-hardness proof of the data complexity of model checking CTL on (non-succinct) one-counter systems. The main technical difficulty is that in a succinct one-counter system of size n there are only $O(n)$ strongly connected components (in the underlying finite automaton describing the system) implying that one can test divisibility only of $O(n)$ many primes.

Theorem 5.12 ([73]) *There exists a fixed* CTL *formula φ such that the following problem is complete for* EXPSPACE*:*

INPUT: A succinct one-counter system $\mathcal{P}$ and a control state q of $\mathcal{P}$.

QUESTION: Does $(\mathcal{T}(\mathcal{P}), q(0)) \models \varphi$ hold? □

5.4 Reachability objectives on one-counter Markov decision processes

Markov decision processes (MDPs) extend classical Markov chains by allowing so called *nondeterministic vertices*. In these vertices, no probability distribution on the outgoing transitions is specified. The other vertices are called *probabilistic vertices*; in these vertices a probability distribution on the outgoing transitions is given. The idea is that in an MDP a player Eve plays against nature (represented by the probabilistic vertices). In each nondeterministic vertex v, Eve chooses a probability distribution on the outgoing transitions of v; this choice may depend on the past of the play (which is a path in the underlying graph ending in v) and is formally represented by a strategy for Eve. An MDP together with a strategy for Eve defines an ordinary Markov chain, whose state space is the unfolding of the graph underlying the MDP.

In this section we consider infinite MDPs, which are finitely represented by one-counter systems; this formalism was introduced in [26] under the name *one-counter Markov decision process* (OC-MDP). For a given OC-MDP $\mathcal{M}$ and a set R of control states of $\mathcal{M}$ (a so called *reachability constraint*) the following two sets $\mathsf{ValOne}(R)$ and $\mathsf{OptValOne}(R)$ were considered in [26]: $\mathsf{ValOne}(R)$ is the set of all states s of the MDP defined by $\mathcal{M}$ such that for every $\varepsilon > 0$ there exists a strategy σ for Eve under which the probability of finally reaching from s a control state in R and at the same time having counter value 0 is at least $1 - \varepsilon$. $\mathsf{OptValOne}(R)$ is the set of all states s of the MDP defined by $\mathcal{M}$ for which there exists a specific strategy for Eve under which this probability becomes 1. It was shown in [26] that for a given OC-MDP $\mathcal{M}$, a set of control states R, and a state s of the MDP defined by $\mathcal{M}$,

- the question whether $s \in \mathsf{OptValOne}(R)$ is PSPACE-hard and in EXP, and
- the question whether $s \in \mathsf{ValOne}(R)$ is hard for every level of the boolean hierarchy BH.

The boolean hierarchy is a hierarchy of complexity classes between NP/coNP and $\mathsf{P}_{\parallel}^{\mathsf{NP}}$ (parallel access to NP) [151]. We use our lower bound techniques (based on the serializability of PSPACE + small depth circuits for converting numbers from Chinese remainder representation to binary representation) in order to improve the second hardness result for the levels of BH to PSPACE-hardness.

Theorem 5.13 ([80]) *Given a OC-MDP a set of its control states R and some control state s it is* PSPACE*-hard to decide whether $s \in \mathsf{ValOne}(R)$.* □

It is worth mentioning that as a byproduct, we also reprove PSPACE-hardness for $\mathsf{OptValOne}(R)$. To the best of the author's knowledge, it is still open whether $\mathsf{ValOne}(R)$ is decidable; the corresponding problem for MDPs defined by pushdown automata is undecidable [64].

5.5 Verification of timed automata

Timed automata were introduced by Alur and Dill [3] and can be seen as an extension of finite automata by allowing the usage of real-time clocks. Timed automata are one of the most important formalisms for modeling real-time systems. In [3] it was shown that the reachability (i.e. emptiness) problem for timed automata is PSPACE-complete. PSPACE-hardness already holds when only three clocks are present as shown by Courcoubetis and Yannakakis [51]. The precise computational complexity of reachability for 2-clock timed automata has been a major open problem only until very recently when Fearnley and Jurdzinksi announced that the problem is indeed PSPACE-hard [65], thus closing the previously best-known lower bound of NP-hardness [124] and the PSPACE upper bound that was known for this problem. It is interesting to note that concerning the reachability problem, there is a close connection between bounded counter automata and timed automata as recently shown by Haase et al. [84]: the reachability problem of n-clock timed automata is equivalent to the the reachability problem of bounded $(n-1)$-counter automata with respect to logarithmic space reductions.

In this section, we present an application of the serializability technique to timed automata. It was shown in [147] that the reachability problem for 2-clock timed automata with modulo tests on counter values is PSPACE-hard. For the lower bound proof in [147] it is crucial that the numerical constants that appear in the transitions of the timed automaton are encoded in binary. We improve the lower bound from [147] by showing that the reachability problem for 2-clock timed automata with modulo tests is already PSPACE-hard when the numbers that occur in transitions are encoded in unary. It shows that very simple extensions of the reachability problem of timed automata with two clocks are PSPACE-hard. In [124] it has been shown that model checking CTL on timed automata with two clocks (but without modulo tests) is PSPACE-hard (and PSPACE-complete). We prove that already the data complexity of this problem is PSPACE-hard, although the very recent PSPACE-hardness result by Fearnley and Jurdzinksi for reachability of 2-clock timed automata [65] implies this. Let us start with the definition of timed automata, see e.g. [22] for more details.

Before we state the main results, let us introduce the model of timed automata. Here, we slightly deviate from the definition of timed automata from [124] (and thus from our journal paper [80]): we do not work with atomic propositons but with atomic actions. Let C be a finite

set, whose elements are called *clocks*. A mapping $t \in \mathbb{R}_+^C$ from C to the set $\mathbb{R}_+$ of positive real numbers is also called a *clock valuation*. The set $B(C)$ of *clock constraints* over C is the set of all boolean formulas with atomic formulas of the form $c \sim k$, where $c \in C$, $k \in \mathbb{N}$ and $\sim \in \{\leq, \geq\}$. We use the usual abbreviations, e.g., we write $c = k$ for $c \leq k \wedge c \geq k$. Let us define the size of the clock constraint $c \sim k$ as $|c \sim k| = \lceil \log k \rceil$; it is the length of the binary encoding of the number k. A clock valuation $t \in \mathbb{R}_+^C$ satisfies a clock constraint $\gamma \in B(C)$, if the formula γ becomes true, when each clock $c \in C$ is replaced by the value $t(c)$.

A *timed automaton* (TA) is a tuple $\mathcal{A} = (Q, \mathsf{A}, C, \delta)$, where

- Q is a finite set of *control states*,
- $\mathsf{A} \subseteq \mathsf{Act}$ is a finite set of action labels,
- C is a finite set of *clocks*, and
- $\delta \subseteq Q \times B(C) \times \mathsf{A} \times 2^C \times Q$ is a finite set of *transitions*.

The *size* of the TA $\mathcal{A}$ is defined as $|\mathcal{A}| = |Q| + |C| + |\mathsf{A}| + \sum_{p \in \mathcal{P}} |Q_p| + \sum_{(p,\gamma,a,R,q) \in \delta} |\gamma|$. A timed automaton $\mathcal{A} = (Q, \mathsf{A}, C, \delta)$ defines a transition system

$$\mathcal{T}(\mathcal{A}) = (Q \times \mathbb{R}_+^C, \mathsf{A} \uplus \{\varepsilon\}, \{\xrightarrow{a} \mid a \in \mathsf{A} \uplus \{\varepsilon\}\}),$$

where $(q,t) \xrightarrow{a} (q',t')$ if, and only if, one of the following two cases holds:

- $a = \varepsilon$, $q = q'$ and there exists $d \in \mathbb{R}_+$ such that $t'(c) = t(c) + d$ for all $c \in C$ (time d elapses).
- $a \in \mathsf{A}$ and there exists a transition $(q, \gamma, a, R, q') \in \delta$ such that (i) the mapping $t : C \to \mathbb{R}_+$ satisfies the clock constraint γ, (ii) $t'(c) = t(c)$ for all $c \in C \setminus R$, and (iii) $t'(c) = 0$ for all $c \in R$ (i.e., all clocks from the set R are reset).

In this section, we will only consider timed automata with only two clocks x and y. For a natural number m let $t_m : \{x, y\} \to \mathbb{R}_+$ be the clock valuation with $t_m(x) = m$ and $t_m(y) = 0$.

CTL model checking on timed automata

As mentioned above in [124], it was shown that model checking CTL over 2-clock timed automata is PSPACE-complete. The proof in [124] for PSPACE-hardness only works if the timed automaton and the CTL formula are part of the input. Here we sharpen this result by showing that model checking CTL over 2-clock timed automata is PSPACE-hard already for a fixed CTL formula.

Proposition 5.14 ([80]) *There exists a fixed* EF *formula $\varphi$for which the following problem can be computed by a logspace transducer:*
INPUT: A boolean formula $F = F(x_1, \ldots, x_n)$.
OUTPUT: A 2-clock TA $\mathcal{A}(F)$ with distinguished control states in *and* out *such that for every number $0 \leq M \leq 2^n - 1$ the following are equivalent:*

- $F(\mathsf{BIN}_n(M)) = 1$.
- *There exists a path from* (in, t_M) *to* (out, t_M) *in* $\mathcal{T}(\mathcal{A}(F))$ *such that* φ *holds on every state of this path.* □

By making use of the previous Proposition we can prove the following PSPACE lower bound on model checking 2-clock TA against a fixed CTL formula.

Theorem 5.15 ([80]) *There exists a fixed* CTL *formula of the form* $\mathsf{E}\varphi_1\mathsf{U}\varphi_2$*, where* φ_1 *and* φ_2 *are* EF *formulas, such that the following problem is* PSPACE*-complete:*
INPUT: A 2-clock TA $\mathcal{A}$ *and a control state* q *of* $\mathcal{A}$*.*
QUESTION: $(\mathcal{T}(\mathcal{A}), (q, t_0)) \models \mathsf{E}\varphi_1\mathsf{U}\varphi_2$*?* □

Reachability of timed automata with modulo tests

The final application of our lower bound technique concerns the control state reachability problem of timed automata with two clocks but very simple modulo tests. The expressiveness of timed automata with periodic clock constraints has already been studied in [44]. We refer to [147], where it has been shown that the control state reachability problem (or equivalently the emptiness problem) for 2-clock timed automata with modulo tests is PSPACE-hard (and in fact PSPACE-complete). However, the lower bound construction in [147] heavily requires the constants appearing in the clock constraints to be presented in *binary*.

The set $\mathrm{Mod}(C)$ of *modulo clock constraints* over a set of clocks C is the set of boolean formulas with atomic formulas of the form $c \equiv k \bmod \ell$ and $c \sim k$, where $c \in C$, $k, \ell \in \mathbb{N}$ and $\sim \in \{\leq, \geq\}$. A *modulo timed automaton (MTA)* is a tuple $\mathcal{A} = (Q, \{Q_p \mid p \in \mathcal{P}\}, C, \delta)$, where everything is the same as for timed automata, but where $\delta \subseteq Q \times \mathrm{Mod}(C) \times \mathsf{A} \times 2^C \times Q$. The *size* $|\mathcal{A}|$ of an MTA $\mathcal{A}$ is defined in analogy to TA. A clock valuation $t : C \to \mathbb{R}_+$ *satisfies* a modulo constraint of the form $c \equiv k \bmod \ell$ whenever $\lfloor t(c) \rfloor \equiv k \bmod \ell$, where for each $r \in \mathbb{R}_+$ we define $\lfloor r \rfloor$ to be the largest non-negative integer n such that $n \leq r$. The transition system $\mathcal{T}(\mathcal{A})$ of an MTA $\mathcal{A}$ is defined analogously as for timed automata (by taking into account the above definition when a clock valuation satisfies a modulo constraint).

Theorem 5.16 ([80]) *The following problem is* PSPACE*-hard, even if all constants that occur in the input are given in unary:*
INPUT: An MTA $\mathcal{A} = (Q, \mathsf{A}, C, \delta)$ *with only two clocks* x *and* y *and two distinguished control states* $q_0, q_1 \in Q$ *such that every transition* $(q, \gamma, a, R, q') \in \delta$ *satisfies*

- γ *does not contain any atomic formulas of the form* $x \sim k$*,*
- $x \notin R$ *(i.e.* x *is never reset),*
- γ *does not contain any atomic formulas of the form* $y \equiv k$ *mod* ℓ*, and*
- *if* $y \sim k$ *is an atomic formula in* γ*, then* $k = 1$*.*

QUESTION: Does $(q_0, t_0) \to^* (q_1, t)$ *hold for some clock valuation* $t \in \mathbb{R}_+^{\{x,y\}}$ *in* $\mathcal{T}(\mathcal{A})$*?* □

6 Model Checking simple logics on one-counter systems

In this section we concern ourselves with model checking HM and EF on one-counter systems and succinct/parametric one-counter systems.

Until recently the best-known upper bound for model checking EF on one-counter systems was PSPACE proven by Serre [169] which already holds for model checking the modal μ-calculus. The previously best-known lower bound for this problem was hardness for DP which was extended to a $\mathsf{P}_{\|}^{\mathsf{NP}}$ lower bound in [72]. In Section 6.1 we show that this problem lies in P^{NP} and is thus P^{NP}-complete by Theorem 5.11. As an application of this upper bound result, we prove that weak bisimilarity between a one-counter system and a finite system is in P^{NP}, and in fact P^{NP}-complete. It is worth mentioning that there are only very few natural problems known to be complete for P^{NP}; the above-mentioned MAX-LEX-SAT being one of them. Moreover, we show that there is a fixed EF formula for which model checking one-counter systems is hard for $\mathsf{P}_{\|}^{\mathsf{NP}}$.

In Section 6.2 we discuss model checking parametric one-counter automata against HM and EF specifications. We show that both model checking HM and EF is PSPACE-complete on succinct one-counter systems. The latter is surprising since there is already a fixed CTL for which model checking succinct one-counter systems is EXPSPACE-hard (Theorem 5.12). We also study the model checking problem for HM and EF on parametric one-counter systems. We prove that model checking EF on parametric one-counter systems is undecidable via reduction from Hilbert's Tenth Problem. Finally, model checking HM on parametric one-counter systems is shown to be PSPACE-complete.

Bibliographic notes. The results on model checking EF one-counter systems have been published in the conference paper [82] (LICS 2009) in joint work with Richard Mayr and Anthony Widjaja To. The results on model checking CTL on succinct and parametric one-counter systems have been published in the conference paper [73] (ICALP 2010) in joint work with Christoph Haase, Joël Ouaknine and James Worrell. The results on model checking HM and EF on succinct and parametric one-counter systems have been published in the conference paper [74] (FOSSACS 2012) in joint work with Christoph Haase, Joël Ouaknine and James Worrell.

6.1 EF model checking on one-counter systems is P^{NP}-complete

In this section we prove that model checking EF on OCS is in P^{NP} and hence P^{NP}-complete by Theorem 5.11. For the upper bound we even restrict ourselves to the case when the input

formulas are given as directed acyclic graphs (DAGs). Firstly, this implies that our result is more general, but more importantly it allows us to reduce in polynomial time the question whether a given one-counter system is bisimilar to a finite transition system to EF model checking on one-counter systems.

We not only prove that the model checking problem as a decision problem is in P^{NP} but rather solve the *global* model checking problem. The global model checking problem is a computation problem and asks to compute for a given transition system $\mathcal{T}$ and a given formula φ a presentation of the set of all states satifying φ. In our case of model checking one-counter systems, such a global presentation will be given in terms of formulas of an adequate Presburger-like logic. In fact, it turned out that only this more general approach allowed us to solve this problem in the end. Our proof strategy is as follows. We define a syntactic variant of Presburger arithmetic that we call MIN-MAX ARITHMETIC for which we prove two things:

- The membership problem (i.e. the question given a formula ψ of MIN-MAX ARITHMETIC and a natural number n to decide whether $n \in [\![\psi]\!]$) for MIN-MAX ARITHMETIC is P^{NP}-complete.
- Given a one-counter system $\mathcal{P}$ with control states Q and an EF formula φ one can compute for each control state $q \in Q$ a formula ψ_q in MIN-MAX ARITHMETIC such that

  $$[\![\psi_q]\!] = \{n \in \mathbb{N} \mid (\mathcal{T}(\mathcal{P}), q(n)) \models \varphi\}.$$

  Thus, for each control state q one can compute a presentation ψ_q of the set of naturals that satisfy the EF formula φ in control state q.

Min-Max Arithmetic

Let us define MIN-MAX ARITHMETIC. Formally, a MIN-MAX ARITHMETIC formula (in DAG presentation is a sequence of definitions $\alpha = (\alpha_i)_{i \in [1,\ell]}$ for some $\ell \geq 1$, where for each $i \in [1, \ell]$ the *definition* α_i is precisely one of the following, where $j, k \in [1, i-1]$ and where $\circ \in \{\leq, \geq\}$:

(1) $\equiv m \bmod n$, where $n > 0$ and $m \in \mathbb{Z}/n\mathbb{Z}$,

(2) $\circ n$, where $n \in \mathbb{N}$,

(3) $\neg \alpha_j$

(4) $\alpha_j \wedge \alpha_k$,

(5) $\circ \min \alpha_j$,

(6) $n \circ \min \alpha_j$, where $n \in \mathbb{N}$,

(7) $\circ \max(\alpha_j, n)$, where $n \in \mathbb{N}$, or

(8) $m \circ \max(\alpha_j, n)$, where $m, n \in \mathbb{N}$.

We formally put $\min \emptyset = \infty$ and $\max \emptyset = -1$. Let us now define the semantics of MIN-MAX ARITHMETIC DAG formulas. For each α_i we define the set $[\![\alpha_i]\!] \subseteq \mathbb{N}$ inductively as follows:

(1) $[\![\equiv m \bmod n]\!] \stackrel{\text{def}}{=} \{k \in \mathbb{N} \mid k \equiv m \bmod n\}$,

(2) $[\![\circ n]\!] \stackrel{\text{def}}{=} \{k \in \mathbb{N} \mid k \circ n\}$,

(3) $[\![\neg \alpha_j]\!] \stackrel{\text{def}}{=} \mathbb{N} \setminus [\![\alpha_j]\!]$,

(4) $[\![\alpha_j \wedge \alpha_k]\!] \stackrel{\text{def}}{=} [\![\alpha_j]\!] \cap [\![\alpha_k]\!]$,

(5) $[\![\circ \min \alpha_j]\!] \stackrel{\text{def}}{=} \{k \in \mathbb{N} \mid k \circ \min[\![\alpha_j]\!]\}$,

(6) $[\![n \circ \min \alpha_j]\!] \stackrel{\text{def}}{=} \begin{cases} \mathbb{N} & \text{if } n \circ \min[\![\alpha_j]\!] \\ \emptyset & \text{otherwise} \end{cases}$,

(7) $[\![\circ \max(\alpha_j, n)]\!] \stackrel{\text{def}}{=} \{k \in \mathbb{N} \mid k \circ \max([\![\alpha_j]\!] \cap [0, n])\}$,

(8)

$$[\![m \circ \max(\alpha_j, n)]\!] \stackrel{\text{def}}{=} \begin{cases} \mathbb{N} & \text{if } m \circ \max([\![\alpha_j]\!] \cap [0, n]) \\ \emptyset & \text{otherwise.} \end{cases}$$

We define $[\![\alpha]\!] \stackrel{\text{def}}{=} [\![\alpha_\ell]\!]$. Observe that MIN-MAX ARITHMETIC formulas can be seen as a fragment of Presburger arithmetic (formulas in one free variable, thus defining ultimately periodic sets) being equi-expressive but with different succinctness.

The *size* of a MIN-MAX ARITHMETIC formula is defined as expected, where each of the occurring constants is presented in binary.

The following lemma states that the set of naturals that is defined by a formula of MIN-MAX ARITHMETIC is ultimatily period with a threshold and period that is computable in polynomial time.

Lemma 6.1 (Periodicity Lemma for MIN-MAX ARITHMETIC, [82]) *Given a* MIN-MAX ARITHMETIC *formula $\alpha = (\alpha_i)_{i \in [1,\ell]}$ one can compute in polynomial time a threshold t_α and a period p_α such that the following holds:*

$$\forall n, n' > t_\alpha : \qquad n \equiv n' \mod p_\alpha \qquad \Rightarrow \qquad n \in [\![\alpha]\!] \Leftrightarrow n' \in [\![\alpha]\!]$$

The previous lemma is central for proving that the membership problem for MIN-MAX ARITHMETIC is in P^{NP} (in fact, one can prove that the membership problem is P^{NP}-complete).

Proposition 6.2 ([82]) *The following problem is P^{NP}-complete:*
INPUT: $n \in \mathbb{N}$ in binary and a MIN-MAX ARITHMETIC *formula α.*
QUESTION: $n \in [\![\alpha]\!]$? □

From EF model checking to Min-Max Arithmetic

For the rest of this section, let us fix some one-counter system $\mathcal{P} = (Q, \mathsf{A}, \delta_0, \delta_{>0})$. For technical reasons, we add a fresh atomic action $\lambda \in \mathsf{A}$ that does not previously occur in $\delta_0 \cup \delta_{>0}$ and which we fix for the rest of this section. Our goal is to "saturate" $\mathcal{P}$ with λ-labeled transitions yielding a new OCS $\mathcal{P}'$ that allows us to characterize the reachability relation in $\mathcal{T}(\mathcal{P})$ in terms of *normalized paths* in $\mathcal{T}(\mathcal{P}')$. By normalized paths we mean paths in which the sequence of counter values of the states in the path are first non-increasing and then non-decreasing. Let $\mathcal{P}'$ denote the resulting one-counter system *after saturation* (to be made more precise below). Our saturation construction has the following motivation:

(1) One can compute in polynomial time all information needed for representing normalized paths in $\mathcal{T}(\mathcal{P}')$ in terms of few small arithmetic progressions (more details below).

(2) For every EF formula φ in which λ does not occur we have $(\mathcal{T}(\mathcal{P}), q(n)) \models \varphi$ if, and only if, $(T(\mathcal{P}'), q(n)) \models \varphi$ for every configuration $q(n)$.

A *path* in $\mathcal{T}(\mathcal{P})$ is a non-empty finite sequence of transitions $\pi = q_1(n_1) \to q_2(n_2) \cdots \to q_k(n_k)$ in $\mathcal{T}(\mathcal{P})$. We call π *mountain*, if $n_1 = n_k$ and $n_i \geq n_1$ for each $i \in [1, k]$. We call π *zero*, if $n_i = 0$ for some $i \in [1, k]$, otherwise we call π *positive*. Let $q_1(n_1), q_2(n_2) \in Q \times \mathbb{N}$ be configurations. Then, we write $q_1(n_1) \downarrow_{\mathcal{T}(\mathcal{P})} q_2(n_2)$ (resp. $q_1(n_1) \uparrow_{\mathcal{T}(\mathcal{P})} q_2(n_2)$) whenever $q_1(n_1) \to q_2(n_2)$ is a transition in $\mathcal{T}(\mathcal{P})$ and $n_2 \leq n_1$ (resp. and $n_2 \geq n_1$) . We now present a saturation construction that allows us to shortcut mountain paths by adding λ-transitions.

Choosing control states $q, q' \in Q$ and $\delta \in \{\delta_0, \delta_{>0}\}$, we now present rules (R1) to (R4) that can be applied only if $(q, \lambda, 0, q') \notin \delta$. In this case, we can add the transition $(q, \lambda, 0, q')$ to δ if at least one of the following conditions holds:

(R1) $(q, a, 0, q') \in \delta$ for some $a \in \mathsf{A}$.

(R2) $(q, a_1, +1, q_1) \in \delta$ and $(q_1, a_2, -1, q') \in \delta_{>0}$ for some $q_1 \in Q$ and some $a_1, a_2 \in \mathsf{A}$.

(R3) $(q, a_1, +1, q_1) \in \delta$, $(q_1, \lambda, 0, q_2) \in \delta_{>0}$, and $(q_2, a_2, -1, q') \in \delta_{>0}$ for some $q_1, q_2 \in Q$ and some $a_1, a_2 \in \mathsf{A}$.

(R4) $(q, \lambda, 0, q_1) \in \delta$ and $(q_1, \lambda, 0, q') \in \delta$ for some $q_1 \in Q$.

Formally, let $\mathcal{P}' = (Q, \mathsf{A}, \delta'_0, \delta'_{>0})$ denote the unique one-counter system that we obtain from $\mathcal{P}$ by applying rules (R1)–(R4) until no longer possible. Clearly, one applies at most $|Q|^2$ such saturation steps in total.

The following lemma characterizes the reachability relation in $\mathcal{T}(\mathcal{P})$ with the one in $\mathcal{T}(\mathcal{P}')$.

Lemma 6.3 ([82]) *Let $p(m), q(n) \in Q \times \mathbb{N}$ be configurations. Then, the following three statements are equivalent:*

(1) $p(m) \to^* q(n)$ holds in $\mathcal{T}(\mathcal{P})$.

(2) $p(m) \to^* q(n)$ holds in $\mathcal{T}(\mathcal{P}')$.

(3) There exists some configuration $r(k) \in Q \times \mathbb{N}$ such that $p(m) \downarrow^*_{\mathcal{T}(\mathcal{P}')} r(k)$ and $r(k) \uparrow^*_{\mathcal{T}(\mathcal{P}')} q(n)$. □

Observe that if $q_1(n_1) \uparrow^*_{\mathcal{T}(\mathcal{P}')} q_2(n_2)$ and $n_1 > 0$, then also $q_1(n_1+i) \uparrow^*_{\mathcal{T}(\mathcal{P}')} q_2(n_2+i)$ for each $i \in \mathbb{N}$. Similarly, if $q_1(n_1) \downarrow^*_{\mathcal{T}(\mathcal{P}')} q_2(n_2)$ and $n_2 > 0$, then also $q_1(n_1+i) \downarrow^*_{\mathcal{T}(\mathcal{P}')} q_2(n_2+i)$ for each $i \in \mathbb{N}$. This motivates us to define, for each $q_1, q_2 \in Q$, the following set of differences of counter values of monotone positive paths:

$$\Delta^{>0}_{\uparrow}(q_1, q_2) \stackrel{\text{def}}{=} \{d \in \mathbb{N} \mid q_1(1) \uparrow^*_{\mathcal{T}(\mathcal{P}')} q_2(d+1)\}$$
$$\Delta^{>0}_{\downarrow}(q_1, q_2) \stackrel{\text{def}}{=} \{d \in \mathbb{N} \mid q_1(d+1) \downarrow^*_{\mathcal{T}(\mathcal{P}')} q_2(1)\}$$

Analogously, we collect the set of differences of counter values of monotone zero paths:

$$\Delta^{=0}_{\uparrow}(q_1, q_2) \stackrel{\text{def}}{=} \{d \in \mathbb{N} \mid q_1(0) \uparrow^*_{\mathcal{T}(\mathcal{P}')} q_2(d)\}$$
$$\Delta^{=0}_{\downarrow}(q_1, q_2) \stackrel{\text{def}}{=} \{d \in \mathbb{N} \mid q_1(d) \downarrow^*_{\mathcal{T}(\mathcal{P}')} q_2(0)\}$$

A theorem due to Chrobak [46] and Martinez [133] states that from a nondeterministic finite automaton over a unary alphabet one can compute in polynomial time an at most quadratically larger equivalent one that is in a certain normal form (Chrobak normal form). However, both papers contain a subtle flaw that was recently fixed in [183]. The proof of the following lemma will make use of this result.

Lemma 6.4 ([82]) *Each of the sets $\Delta^{>0}_{\uparrow}(q_1, q_2)$, $\Delta^{>0}_{\downarrow}(q_1, q_2)$, $\Delta^{=0}_{\uparrow}(q_1, q_2)$, $\Delta^{=0}_{\downarrow}(q_1, q_2)$ is equivalent to a union of $O(|Q|^2)$ arithmetic progressions with offsets bounded by $O(|Q|^2)$ and periods bounded by $O(|Q|)$ that are moreover computable in polynomial time.* □

Let $q_1, q_2 \in Q$ be control states. Note that if $q_1(n) \downarrow^*_{\mathcal{T}(\mathcal{P}')} q_3(1) \uparrow^*_{\mathcal{T}(\mathcal{P}')} q_2(n)$ for some $q_3 \in Q$, then also $q_1(n+i) \downarrow^*_{\mathcal{T}(\mathcal{P}')} q_3(1+i) \uparrow^*_{\mathcal{T}(\mathcal{P}')} q_2(n+i)$. Therefore, we define $\nabla(q_1, q_2) \in \mathbb{N} \cup \{\infty\}$ to be

$$\min\{n > 0 \mid \exists q_3 \in Q : q_1(n) \downarrow^*_{\mathcal{T}(\mathcal{P}')} q_3(1) \uparrow^*_{\mathcal{T}(\mathcal{P}')} q_2(n)\}.$$

Observe that $\nabla(q, q) = 1$ for every $q \in Q$.

Lemma 6.5 ([82]) *Either $\nabla(q_1, q_2) = \infty$ or $\nabla \in O(|Q|^2)$. Moreover $\nabla(q_1, q_2)$ can be computed in polynomial time.* □

The next lemma characterizes zero paths.

Lemma 6.6 ([82]) *There is a zero path from $q(n)$ to $q'(n')$ in $\mathcal{T}(\mathcal{P}')$ if and only if $n \in \Delta^{=0}_{\downarrow}(q, q'')$ and $n' \in \Delta^{=0}_{\uparrow}(q'', q')$ for some $q'' \in Q$.* □

The next lemma characterizes positive paths.

Lemma 6.7 ([82]) *Assume $n \leq n'$. Then there exists a positive path in $\mathcal{T}(\mathcal{P}')$ from $q(n)$ to $q'(n')$ if and only if $n \geq \nabla(q, q'')$ and $n' - n \in \Delta^{>0}_{\uparrow}(q'', q')$ for some $q'' \in Q$.*
Assume $n \geq n'$. Then there exists a positive path from (q, n) to (q', n') in $\mathcal{T}(\mathcal{P}')$ if and only if $n' \geq \nabla(q'', q')$ and $n - n' \in \Delta^{>0}_{\downarrow}(q, q'')$ for some $q'' \in Q$. □

Lemma 6.6 and Lemma 6.7 are the central ingredients to compute, for each control state, a presentation of the natural numbers that satisfy a given EF formula in terms of MIN-MAX ARITHMETIC

Theorem 6.8 ([82]) *From a given one-counter system $\mathcal{P}$ and a given* EF *formula φ, one can compute in polynomial time for each control state q of $\mathcal{P}$ an* MIN-MAX ARITHMETIC *formula α_q such that $[\![\alpha_q]\!] = \{n \in \mathbb{N} \mid (\mathcal{T}(\mathcal{P}), q(n)) \models \varphi\}$.* □

By combining Theorem 6.8 with Proposition 6.2 we obtain the following corollary.

Corollary 6.9 ([82]) *The following problem is in P^{NP}:*
INPUT: A one-counter system $\mathcal{P} = (Q, \mathsf{A}, \delta_0, \delta_{>0})$, a configuration $q(n) \in Q \times \mathbb{N}$ and an EF *formula φ.*
QUESTION: $(\mathcal{T}(\mathcal{P}), q(n)) \models \varphi$? □

Since one can reduce in polynomial time weak bisimilarity of given one-counter systems against a given finite transition system to model checking one-counter systems against EF formulas in DAG representation [122, 121], we obtain the following corollary.

Corollary 6.10 ([82]) *One can decide in P^{NP} whether a given one-counter system is weakly bisimilar to a given finite transition system.* □

Some lower bounds

The data complexity of model checking EF on one-counter systems can shown to be P^{NP}-hard by a reduction from the problem INDEX-ODD, which asks for a given list $\varphi_1, \ldots, \varphi_n$ of boolean formulas, whether there exists an odd index i such that $\varphi_1, \ldots, \varphi_i$ are all satisfiable and $\varphi_{i+1}, \ldots, \varphi_n$ are all unsatisfiable.

Theorem 6.11 ([82]) *There exists a fixed* EF *formula for which model checking a given one-counter system is $\mathsf{P}^{\mathsf{NP}}_{\parallel}$-hard.* □

For the latter problem the best-known upper bound is P^{NP}, so there still remains a complexity gap for this problem; but recall that for the combined complexity we have a matching P^{NP} lower bound by Theorem 5.11. The latter also applies for weak bisimilarity against finite systems by the following theorem proven by a reduction from the P^{NP}-complete problem DSAT from [151], thus matching Corollary 6.10.

Theorem 6.12 ([82]) *Deciding whether a given one-counter system is weakly bisimilar to a given finite transition system is P^{NP}-hard.* □

6.2 Model checking succinct and parametric one-counter systems

Before we discuss our main results on model checking succinct and parametric one-counter systems, let us define them more formally. Let $X = \{x_1, \ldots, x_n\}$ denote a finite set of *parameters*,

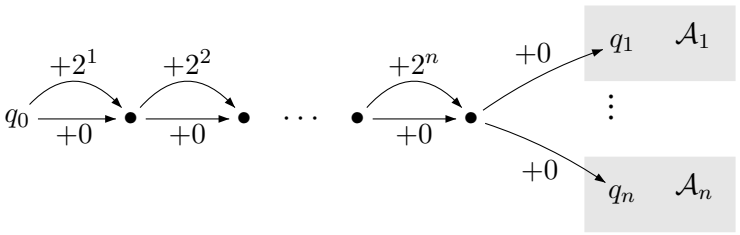


Figure 6.1: A part of a succinct one-counter system $\mathcal{A}$ constructed for simulating the validity of a QBF formula.

and let $\mathsf{Op} \stackrel{\text{def}}{=} \{\mathsf{add}(z), \mathsf{add}(x) : z \in \mathbb{Z}, x \in X\} \cup \{\mathsf{zero}\}$ be a set of *operations*. A *parametric one-counter system* is a tuple $\mathcal{A} = (Q, \mathsf{A}, X, \Delta)$, where Q is a finite set of *control states*, and $\Delta \subseteq Q \times \mathcal{A} \times \mathsf{Op} \times Q$ is the *transition relation*. A *succinct one-counter system* is a parametric one-counter system with $X = \emptyset$. We write $q \xrightarrow{a,\mathsf{op}} q'$ whenever $(q, a, \mathsf{op}, q') \in \Delta$ or just $q \xrightarrow{\mathsf{op}} q'$ when the action a is not important. A *valuation* $\nu : X \to \mathbb{Z}$ is a function assigning an integer to each parameter. Given a parametric one-counter system $\mathcal{A}$, a valuation induces a succinct one-counter system $\mathcal{A}^\nu$ which is obtained by replacing each transition $q \xrightarrow{a,\mathsf{add}(x_i)} q'$ by $q \xrightarrow{a,\mathsf{add}(\nu(x_i))} q'$. For a succinct one-counter system $\mathcal{A}$, we define its underlying transition system as $\mathcal{T}(\mathcal{A}) \stackrel{\text{def}}{=} (Q \times \mathbb{N}, \mathsf{A}, \{\xrightarrow{a} | \ a \in \mathsf{A}\})$, where $q(n) \xrightarrow{a} q'(n')$ if, and only if, either $q \xrightarrow{\mathsf{add}(z)} q'$ and $n' = n + z$, or $q \xrightarrow{a,\mathsf{zero}} q' \in \Delta$ and $n = n' = 0$.

The *model-checking problem* for parametric (and thus for succinct) one-counter systems is defined as follows.

MODEL CHECKING FOR PARAMETRIC ONE-COUNTER SYSTEMS

INPUT: A parametric one-counter system $\mathcal{A}$ with control states Q, $q \in Q$ and a formula φ.

QUESTION: Does $(T(\mathcal{A}^\nu), q(0)) \models \varphi$ hold for each assignment $\nu : X \to \mathbb{Z}$?

Model checking succinct one-counter systems

In this section we prove that model checking HM succinct one-counter systems is PSPACE-hard and that model checking EF on them is in PSPACE. Thus both model checking HM and EF on succinct one-counter systems turns out to be PSPACE-complete. These results can be seen in stark contrast to EXPSPACE-completeness of model checking CTL on succinct one-counter systems (Theorem 5.12).

The following proposition on PSPACE-hardness of model checking HM on one-counter systems can easily proven by a reduction from the validity problem quantified Boolean formulas (QBF): A part of the succinct one-counter system that one constructs in the reduction is depicted in Figure 6.1. We do not go into further proof details here.

Proposition 6.13 ([74]) *Model checking* HM *on succinct one-counter systems is* PSPACE*-hard.*□

Next we are going to outline the proof that EF model checking on succinct one-counter systems is in PSPACE, and hence PSPACE-complete by Proposition 6.13. To this end, let us fix some succinct one-counter system $\mathcal{A}$ with control states Q. Our result is based on the following combinatorial lemma, which expresses periodicity properties of reachability relations in the succinct one-counter automaton $\mathcal{A}$.

Lemma 6.14 ([74]) *There are naturals $\tau, \delta \leq \mathsf{exp}(|\mathcal{A}|)$ such that for each $n, n', m, m' > \tau$ with $n \equiv n'$ mod δ and $m \equiv m'$ mod δ the following statements hold for each $q, q' \in Q$:*

(1) $q(n) \longrightarrow^* q'(m)$ if, and only if, $q(n') \longrightarrow^* q'(m')$ in $\mathcal{T}(\mathcal{P})$.

(2) $q(n) \longrightarrow^*_{\mathcal{A}} q'(m)$ if, and only if, $q(n') \longrightarrow^*_{\mathcal{A}} q'(m')$ in $\mathcal{T}(\mathcal{P})$. □

Let us assume the values τ and δ from Lemma 6.14 to be fixed for the rest of this section. For the PSPACE upper bound, we will show that $\{n \in \mathbb{N} \mid (\mathcal{T}(\mathcal{P}), q(n)) \models \varphi\}$ is ultimately periodic with periodicity δ (the periodicity is thus independent from $|\varphi|$).

Lemma 6.15 ([74]) *We have $(\mathcal{T}(\mathcal{P}), q(n)) \models \varphi$ if, and only if, $(\mathcal{T}(\mathcal{P}), q(n')) \models \varphi$ for each control state $q \in Q$ and each* EF *formula φ, provided $n, n' > \tau + \delta \cdot |\varphi|$ and $n \equiv n'$ mod δ.* □

The previous lemma can now directly used to construct an alternating polynomial time algorithm for model checking EF on one-counter systems: as an important blackbox tool it uses a result from [83] that states that reachability for succinct (even parametric) one-counter systems is in NP.

Theorem 6.16 ([74]) EF *model checking of succinct one-counter systems is in* PSPACE. □

Model checking parametric one-counter systems

For the rest of this section we concern ourselves with model checking parametric one-counter systems.

First, we consider model checking EF on parametric one-counter systems and show that this problem is Π^0_1-complete. With EF being a notational fragment of CTL, membership in Π^0_1 follows from the very simple fact that CTL model checking on parametric one-counter systems is in Π^0_1 [73]. Thus, we concentrate in this section on a matching Π^0_1-lower bound by giving a reduction from Hilbert's Tenth Problem to the complement of the model checking problem.

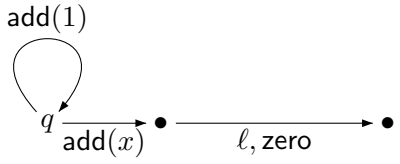
HILBERT'S TENTH PROBLEM (HTP)

INPUT: A polynomial p with coefficients ranging over the integers.
QUESTION: Do there exist $a_1, \ldots, a_n \in \mathbb{Z}$ such that $p(a_1, \ldots, a_n) = 0$?

HTP was shown to be Σ^0_1-complete by Matiyasevich [135]. Note that HTP remains Σ^0_1-hard if we restrict the a_i to range over $\mathbb{N}$: A Diophantine equation $p(x_1, x_2, .., x_n) = 0$ is solvable in the integers if, and only if, one of the 2^n equations $p(\pm x_1, \ldots, \pm x_n) = 0$ has a solution in the naturals. Replacing every unknown with the sum of squares of four unknowns gives, by Lagrange's Theorem, the reduction in the other direction.

Moreover, we may assume without loss of generality that $a_i > 0$ for each $i \in [1, n]$. If some a_i were to be zero in a solution, we can obtain a new polynomial p' in $n-1$ variables by replacing a_i with 0 in p.

Let us fix some polynomial p with coefficients ranging over $\mathbb{Z}$. We will subsequently show how we can compute from p a parametric one-counter system $\mathcal{A}_p$ with a control state q_p and an EF formula φ_p such that p has a solutions over the naturals if, and only if, $(\mathcal{T}(\mathcal{A}_p^\nu), q_p(0)) \models \varphi_p$ for *some* valuation ν of the parameters of $\mathcal{A}$. Recall that the valuation of the parameters of $\mathcal{A}_p$ ranges over $\mathbb{Z}$. However, we can easily ensure with a simple EF formula that a parameter x is positive. For the following succinct one-counter system $\mathcal{A}_{x\geq 1}$

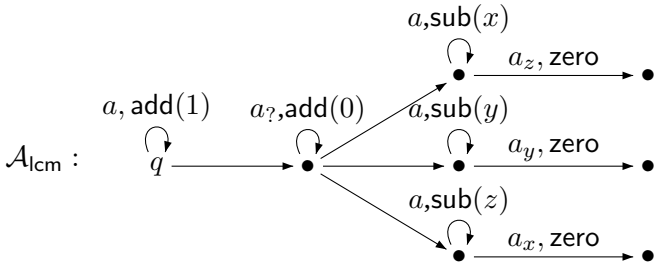


we have $(\mathcal{T}(\mathcal{A}_{x\geq 1}^\nu), q(0)) \models \neg \mathsf{EF}\langle \ell \rangle \mathtt{true}$ if, and only if, $\nu(x) \geq 1$.

More challenging than testing if a parameter is positive when reducing from HTP is that we need to be able to express a multiplication relation over the parameters in the parametric one-counter system. In order to do that, we employ a trick that became popular by the work of Robinson [158] which allows us to define multiplication in terms of the least common multiple. In fact given $x, y \in \mathbb{N}$, we have

$$\begin{aligned} &\mathsf{lcm}(x+y, x+y+1) - \mathsf{lcm}(x, x+1) - \mathsf{lcm}(y, y+1) \\ =&(x^2 + x + 2xy + y^2 + y) - (x^2 + x) - (y^2 + y) \quad = \quad 2xy \end{aligned}$$

We note that addition and subtraction of the parameters can easily be realized by introducing additional slack parameters in the parametric one-counter system. Thus, we can enhance our parametric one-counter system by transitions of the kind $\mathsf{sub}(x)$, meaning that $\nu(x)$ is subtracted from the counter, provided the counter is at least $\nu(x)$. We now demonstrate that for parameters x, y, z of some parametric one-counter system that each assume positive values, which we can check as seen above, we can "express" in EF that $z = \mathsf{lcm}(x, y)$. Consider the following parametric one-counter system $\mathcal{A}_{\mathsf{lcm}}$, where unlabeled transitions are assumed to be labeled with "$a, \mathsf{add}(0)$":



The idea is to express that for all $n \in \mathbb{N}$, we have that both x and y divide n if, and only if, z divides n. We note that for each $\nu : \{x, y, z\} \to \mathbb{Z}$ with $\nu(x), \nu(y), \nu(y) \geq 1$ we have that $(\mathcal{T}(\mathcal{A}_{\mathsf{lcm}}^\nu), q(0))) \models \mathsf{AG}(\langle a_? \rangle \mathtt{true} \to ((\mathsf{EF}\langle a_x \rangle \mathtt{true} \wedge \mathsf{EF}\langle a_y \rangle \mathtt{true}) \leftrightarrow \mathsf{EF}\langle a_z \rangle \mathtt{true}))$ if, and only if, $\nu(z) = \mathsf{lcm}(\nu(x), \nu(y))$.

Thus, by introducing a sufficient number of slack variables, we can express multiplication, addition and subtraction, which allows us to solve HTP for any arbitrary polynomial. Thus, we obtain the following theorem.

Theorem 6.17 ([74]) *Model checking* EF *on parametric one-counter systems is* Π_1^0*-complete.*□

We note that by [135] there exists a ***fixed universal*** polynomial $p_u(n, k, x_1, \ldots, x_m)$ such that for each recursively enumerable set $S \subseteq \mathbb{N}$, there is some $k_0 \in \mathbb{N}$ such that $S = \{n \in \mathbb{N} \mid \exists n_1, \ldots, n_m \in \mathbb{N} : p_u(n, k_0, n_1, \ldots, n_m) = 0\}$. This allows us to strengthen our result insofar as there exists a ***fixed*** EF formula φ and a ***fixed*** parametric one-counter system $\mathcal{A}$with a control state $q_0 \in Q$ such that it is Π_1^0-complete to decide for a given $n \in \mathbb{N}$ whether $(T(\mathcal{A}^\nu), q_0(n)) \models \varphi$ holds for all $\nu : X \to \mathbb{Z}$.

The rest of this section will be devoted to sketching a PSPACE upper bound for model checking HM on parametric one-counter systems. Let us fix some parametric one-counter system $\mathcal{A} = (Q, \mathsf{A}, X, \Delta)$ with $X = \{x_1, \ldots, x_\ell\}$, some control state $q_0 \in Q$ and some HM formula α. Since we have already proven that HM model checking of succinct one-counter systems is in PSPACE Theorem 6.16 (we show that even model checking EF on succinct one-counter systems is in PSPACE), in order to obtain a PSPACE upper bound, it is sufficient to show that if $(\mathcal{T}(\mathcal{A}^\nu), q_0(0)) \models \alpha$ holds for some $\nu : X \to \mathbb{Z}$ then there is some $\mu : X \to \mathbb{Z}$ such that $(\mathcal{T}(\mathcal{A}^\mu), q(0)) \models \alpha$ and $|\mu(x)|$ can be represented with polynomially many bits in $|\mathcal{A}| + |\alpha|$ for each $x \in X$, since such an assignment can be guessed in PSPACE.

For each $q \in Q$ and each subformula φ of α, let us define $\mathcal{M}(q, \varphi) \subseteq \mathbb{Z}^\ell \times \mathbb{N} \subseteq \mathbb{Z}^{\ell+1}$ as follows:

$$\mathcal{M}(q, \varphi) \stackrel{\text{def}}{=} \{(z_1, \ldots, z_\ell, n) \mid (\mathcal{T}(\mathcal{A}^\nu), q(n)) \models \varphi \text{ and } \nu(x_i) = z_i, i \in [1, \ell]\}.$$

Before we proceed with the upper bound, we need to introduce some additional notation. For an integer matrix $A = (a_{ij}) \in \mathbb{Z}^{m \times n}$, we denote by $\|A\| = \max_i\{\sum_j |a_{ij}|\}$ ***the norm of*** A. For an integer vector $\vec{b} = (b_i)$, we denote by $\|\vec{b}\| = \sum_i |b_i|$ ***the norm of*** $\vec{b}$. A ***system of linear Diophantine inequalities (SLDI)*** is a system of the form $\mathcal{S} = (A\vec{x} \geq \vec{b})$, where $A \in \mathbb{Z}^{m \times n}$ is an $m \times n$ matrix, $\vec{b} \in \mathbb{Z}^m$ is an m-vector and $\vec{x}$ is an n-vector of indeterminants all ranging over the integers. By $\mathsf{Sol}(\mathcal{S})$, we denote the set of ***integer solutions*** to the SLDI $\mathcal{S} = (A\vec{x} \geq \vec{b})$. Finally, we define $\|\mathcal{S}\|_{\text{mat}} \stackrel{\text{def}}{=} \|A\|$ and $\|\mathcal{S}\|_{\text{vec}} \stackrel{\text{def}}{=} \|\vec{b}\|$.

Recall that $x_1, \ldots, x_\ell$ are the parameters of $\mathcal{A}$. Our overall goal is to express $\mathcal{M}(q, \varphi)$ by a ***union*** of solutions to SLDIs, each of the form

$$\mathcal{S} = (A\vec{x} \geq \vec{b}), \qquad \text{where } A \in \mathbb{Z}^{m \times (\ell+1)} \text{ and } \vec{b} \in \mathbb{Z}^m \text{ for some } m \geq 1.$$

In the remainder of this section, we will assume for any $(A\vec{x} \geq \vec{b})$ that A is some $m \times (\ell + 1)$ matrix and $\vec{b}$ is some m-vector for some $m \geq 1$. The intuition is that the i^{th} component of $\vec{x}$ with $i \in \{1, \ldots, \ell\}$ is going to correspond to the parameter x_i of $\mathcal{A}$ and the $(\ell+1)^{\text{th}}$ component of $\vec{x}$ is going to correspond to the counter value where the HM formula is evaluated. In case $A = (a_{ij})$ we define $\|A\|_{\ell+1} \stackrel{\text{def}}{=} \max\{|a_{i(\ell+1)}| : i \in [m]\}$ and lift this definition to $\|\mathcal{S}\|_{\ell+1} \stackrel{\text{def}}{=} \|A\|_{\ell+1}$.

In order to prove that small valuations $\nu : X \to \mathbb{Z}$ suffice for α, one can prove that for each $q \in Q$ and each subformula φ of α, we have

$$\mathcal{M}(q, \alpha) = \bigcup_{i \in I} \mathsf{Sol}(\mathcal{S}_i) \tag{6.1}$$

for some index set I with $\|\mathcal{S}_i\|_{\text{mat}} \leq \mathsf{poly}(|\varphi|)$ and $\|\mathcal{S}_i\|_{\text{vec}} \leq \mathsf{poly}(|\varphi|) \cdot \mathsf{exp}(|\mathcal{A}|)$ for each $i \in I$. Once this fact has been established, one can show that each SLDI $\mathcal{S}_i$ admits solutions that can be represented using polynomially many bits in $|\mathcal{A}| + |\alpha|$, thus establishing the desired upper bound on necessary valuations of the parameters of $\mathcal{A}$.

We require some additional notation that, together with the subsequent lemma, will be useful for proving the existence of sets of SLDIs of "small" size for each $\mathcal{M}(q, \varphi)$. Let $H \subseteq \mathbb{Z}^{\ell+1}$. We define $H - x_k \stackrel{\text{def}}{=} \{(z_1, \ldots, z_\ell, z_{\ell+1} - z_k) \in \mathbb{Z}^{\ell+1} \mid (z_1, \ldots, z_{\ell+1}) \in H\}$ for each $k \in [\ell]$ and $H - z \stackrel{\text{def}}{=} \{(z_1, \ldots, z_\ell, z_{\ell+1} - z) \in \mathbb{Z}^{\ell+1} \mid (z_1, \ldots, z_{\ell+1}) \in H\}$ for each $z \in \mathbb{Z}$. The following lemma states that solutions to SLDIs are closed under the operations $-x_k$ and $-z$ and gives bounds on the blow-up of the introduced norms. We remark that we do not require an effective variant of this lemma to establish our PSPACE upper bound.

Lemma 6.18 ([74]) *Let $\mathcal{S} = (A\vec{x} \geq \vec{b})$ be an SLDI with $A = (a_{ij}) \in \mathbb{Z}^{m \times (\ell+1)}$. Then the following holds:*

(1) For each $k \in [1, \ell]$ there is some SLDI $\mathcal{S}'$ with $\mathsf{Sol}(\mathcal{S}') = \mathsf{Sol}(\mathcal{S}) - x_k$, $\|\mathcal{S}'\|_{mat} \leq \|\mathcal{S}\|_{mat} + \|\mathcal{S}\|_{\ell+1}$, $\|\mathcal{S}'\|_{\ell+1} = \|\mathcal{S}\|_{\ell+1}$, and $\|\mathcal{S}'\|_{vec} = \|\mathcal{S}\|_{vec}$.

(2) For each $z \in \mathbb{Z}$, there is some SLDI $\mathcal{S}'$ with $\mathsf{Sol}(\mathcal{S}') = \mathsf{Sol}(\mathcal{S}) - z$, $\|\mathcal{S}'\|_{mat} = \|\mathcal{S}\|_{mat}$, $\|\mathcal{S}'\|_{\ell+1} = \|\mathcal{S}\|_{\ell+1}$, and $\|\mathcal{S}'\|_{vec} \leq \|\mathcal{S}\|_{vec} + \|\mathcal{S}\|_{\ell+1} \cdot |z|$. □

The previous lemma allows one to prove (6.1). By $n_{max}(\mathcal{A})$ we denote the largest absolute value of constants appearing in $\mathcal{A}$. The following lemma implies (6.1) and can be proven by induction on the structure of the HM formula.

Lemma 6.19 ([74]) *For every $q \in Q$ and every subformula φ of α in negation normal form, we have $\mathcal{M}(q, \varphi) = \bigcup_{i \in I} \mathsf{Sol}(\mathcal{S}_i)$, where I is some index set and each $\mathcal{S}_i$ is some SLDI with $\|\mathcal{S}_i\|_{mat} \leq |\varphi|$, $\|\mathcal{S}_i\|_{\ell+1} \leq 1$, $\|\mathcal{S}_i\|_{vec} \leq (n_{max}(\mathcal{A}) + 1) \cdot |\varphi|$.* □

The following lemma from [163] states that solvable SLDIs have small solutions whose norm is independent on the number of rows of the SLDI.

Lemma 6.20 ([163], p. 239) *Each solvable SLDI $A\vec{x} \geq \vec{b}$ has a solution of norm at most $\mathsf{poly}(\|A\| + \|\vec{b}\|)$.* □

Let us come back to our original formula α. By Lemma 6.19, there exists some SLDI $\mathcal{S}_i$ such that $\mathcal{M}(q_0, \alpha) = \mathsf{Sol}(\mathcal{S}_i)$, and where $\|\mathcal{S}_i\|_{\text{mat}} \leq |\alpha|$ and $\|\mathcal{S}_i\|_{\text{vec}} \leq (n_{max}(\mathcal{A}) + 1) \cdot |\alpha|$. Since we are interested in whether $(T(\mathcal{A}^\nu), q_0(0)) \models \alpha$ for some $\nu : X \to \mathbb{Z}$, think of adding to each matrix that occurs in $\mathcal{S}_i$ two more rows expressing that $x_{\ell+1} = 0$. Let us call the resulting SLDI $\mathcal{S}_i'$. By Lemma 6.20, we know that if $\mathcal{S}_i'$ is solvable, then $\mathcal{S}_i'$ has a solution of norm at

most $\mathsf{poly}(n_{max}(\mathcal{A}) + |\alpha|)$. In other words, if $(\mathcal{T}(\mathcal{A}^\nu), q_0(0)) \models \alpha$ for some $\nu : X \to \mathbb{Z}$, then $(\mathcal{T}(\mathcal{A}^\mu), q_0(0)) \models \alpha$ already holds for some $\mu : X \to \mathbb{Z}$ and the number of bits for representing $\mu(x)$ is polynomially bounded in $|\mathcal{A}| + |\alpha|$ for each $x \in X$.

Hence, we obtain the following theorem.

Theorem 6.21 ([74]) *Model checking* HM *on parametric one-counter systems is in* PSPACE.□

7 Lower bounds on verifying asynchronous products and the size of Feferman-Vaught decompositions

Concurrent systems are systems which consist of multiple processes that are simultaneously executed and possibly interacting with each other. A standard way of designing concurrent systems is to compose together several individual processes by taking some "product" operators. Various product operators have been introduced in concurrency theory and verification ranging from synchronized products (the strongest form of products) to asynchronous products (the weakest form of products). From the point of view of system design, synchronized products are the most suitable form of compositional operators. Unfortunately, from the point of view of system verification, they are known to be too powerful. For example, while reachability is NL-complete for finite transition systems, it becomes PSPACE-complete when the same problem is considered over synchronized products of finite transition systems (a.k.a. communicating finite-state machines). In the case of infinite-state systems, we see a more drastic change: while reachability is decidable in polynomial time for pushdown systems (PDS), the same problem becomes undecidable when considered over synchronized products of two PDS (note: these subsume Minsky's counter machines).

In order to circumvent the problem of high complexity and undecidability in verifying concurrent systems composed from individual processes via synchronized products, various weaker notions of products were introduced. Apart from asynchronous products which prohibit the processes to communicate, stronger product operators were introduced by restricting the types of synchronization that are allowed among the processes. Several such restrictions include bounded context switches [155], and finite synchronization [199]. These restricted product operators can serve as good underapproximations of synchronized products. For example, a recent study of concurrency bugs conducted by the authors of [129] reveal that many real-world concurrency bugs can be detected within a small number of context switches. In addition, such restrictions also lead to decidability or lower computational complexity in model checking. For example, checking reachability over communicating finite-state machines and communicating pushdown systems with bounded context switches are both NP-complete [155].

When we consider logic model checking, the situation is not as simple. Asynchronous products do not make model checking easier than synchronized products when we use logics like LTL and CTL (and, in fact, even their restrictions to $\mathsf{LTL}(\mathbf{F}_s, \mathbf{G}_s)$ and the logic EG). Intuitively, the reason is that synchronization is easily to simulate in such logics. Consequently, reachability of 2-stack pushdown systems, which is well-known to be undecidable, easily reduces to model checking any of aforementioned logics over asynchronous products of two PDS. In contrast, the situation is substantially better when we consider simpler logics like Hennessy-Milner Logic

(HM) and its extension with the reachability operator (EF). In fact, powerful the *(Feferman-Vaught) compositional method* (e.g. [132, 156, 199]), which reduces model checking of product structures to model checking of their components, can be used for obtaining decidability or better upper complexity bounds of model checking HM and EF. We now state a simplified variant (in comparison to [156]) of the compositional method, where for transition systems $\mathcal{T}_1, \ldots, \mathcal{T}_k$ we denote by $\prod_{i=1}^{k} \mathcal{T}_i$ the asynchronous product of the $\mathcal{T}_i$.

Theorem 7.1 ([156]) *For each* HM/EF *formula φ over the action labels $\mathsf{A} = \mathsf{A}_1 \cup \ldots \cup \mathsf{A}_k$, for nonempty and pairwise disjoint sets $\mathsf{A}_1, \ldots, \mathsf{A}_k$, one can compute k finite sets of* HM/EF *formulas $\{\psi_i^1\}_{i \in I_1}, \ldots, \{\psi_i^k\}_{i \in I_k}$ over $\mathsf{A}_1, \ldots, \mathsf{A}_k$ respectively, and a positive boolean formula (i.e. no negations) β with variables $\{x_i^1\}_{i \in I_1}, \ldots, \{x_i^k\}_{i \in I_k}$ such that for* all *transition systems $\mathcal{T}_1, \ldots, \mathcal{T}_k$ with initial states $s_1, \ldots, s_k$ we have $(\prod_{i=1}^{k} \mathcal{T}_i, \bar{s}) \models \varphi$ if, and only if, $\beta[\mu]$ is true, where $\bar{s} = (s_1, \ldots, s_k)$ and μ assigns the variables of β as follows: $\mu(x_i^j) = 1$ if, and only if, $(\mathcal{T}_j, s_j) \models \psi_i^j$.*

Actually, a stronger version of Theorem 7.1 was proven in [156] (e.g. with atomic propositions). In the statement of Theorem 7.1, the k sets of formulas and the positive boolean formula β are referred to as the *decomposition* of φ. To give some concrete illustrations of the power of this compositional theorem, Theorem 7.1 can be used to show that model-checking *fixed* EF formulas (i.e. the complexity is only measured the size of the system) is NL-complete for the asynchronous product of finite systems (cf. PSPACE-completeness of communicating finite-state systems), PSPACE-complete for the asynchronous product of pushdown systems [182], and P-complete for the asynchronous product of basic process algebras.

Despite the aforementioned usefulness of the compositional method, the technique yields output decompositions with nonelementary complexity in the size of the formula (see [156]), which is not desirable from both theoretical and practical viewpoints. In fact, it was recently shown that when we consider stronger logics like first-order logic, where the compositional method is also possible (e.g. see [132]), this nonelementary complexity is unavoidable [57]. It is natural to ask whether the size of the decomposition is nonelementary when we consider simpler logics like HM or EF. In fact, this open question has been posed in the literature (e.g. [66, 156]). It is worth mentioning that such a nonelementary lower bound on the size of decompositions is simpler to prove for first-order logic, simply because one can enforce models of nonelementary outdegree.

This open question actually brings us to a more fundamental open question: how does the asynchronous product affect the complexity of model checking of HM logic and EF? This open question has manifested itself in the literature in various concrete forms. As an example, take the result that model checking EF over pushdown systems is PSPACE-complete [196]. Over the asynchronous product of *two* pushdown systems, the best algorithm for model checking EF runs in nonelementary time [182]. In fact, the same nonelementary gap is currently present for asynchronous product of two basic process algebras. Failing to answer this open question is also the reason for the existing nonelementary complexity gaps for several verification problems for PA processes [139]. Recall that PA can be seen as the asynchronous product extension of BPA.

In this chapter, we provide answers to the above open questions. A main contribution of this chapter is to show that, for each integer $k > 0$, there exists an asynchronous product of two

basic process algebras whose EF-logic theory requires k-fold exponential time to solve. This means that model checking EF over the class of asynchronous products of two BPA requires nonelementary time, which is in stark contrast to PSPACE-completeness of EF model checking over BPA [196]. As an upshot of our result, it follows that model checking EF-logic over PA-processes requires nonelementary time, which solves an open question posed by Mayr [139] and at the same time a question by Löding on model checking EF on ground tree rewrite systems [128] (the asynchronous product of two BPAs is a very restricted ground tree rewrite system).

We also show that similar results hold for HM. More precisely, we prove that for each integer $k > 0$ there exists an asynchronous product of two *prefix-recognizable systems* (an extension of BPA and PDS introduced by Caucal [40] by allowing infinitely many rewrite rules compactly represented by regular languages) whose HM theory requires k-fold exponential time to solve. This means that model checking HM over the class of asynchronous products of two prefix-recognizable systems requires nonelementary time, which is in stark contrast to PSPACE-completeness of HM model checking over prefix-recognizable systems (which easily follows[1] from the result of [196]).

An important corollary of our two aforementioned results is that there is no elementary algorithm for computing decompositions of formulas in HM and EF in the sense of Theorem 7.1. We even go one step further and show that no decompositions of formulas in HM-logic and EF-logic of elementary size even exist in general (it could still be the case that the decompositions are generally elementarily large only but the algorithms computing them run in nonelementary complexity — but we show that this cannot be the case). In other words, *both* descriptional and computational complexity of the compositional method for HM and EF in the sense of Theorem 7.1 are inherently nonelementary. Incidentally, this also entails the same nonelementary lower bounds for the compositional method provided in [66] since they generalize Theorem 7.1. Wrapping up, our results entail that the compositional method for EF (resp. HM) *inherently* has to output nonelementary big decompositions for asynchronous product and BPAs (resp. prefix-recognizable systems) are classes of "hard instance" (infinite-state) transition systems that witness this.

So far, our nonelementary lower bounds for the compositional method for HM-logic and EF-logic require the use of infinite-state systems. This still leaves the possibility that Theorem 7.1 could hold when we restrict the transition systems under consideration to be finite-state. Questions of this form are of particular interests in finite model theory (e.g. see [126]) and in verification of finite-state systems. We show, however, that the same nonelementary lower bounds even relativize to the class of asynchronous products of finite systems.

This chapter is organized as follows. We fix notations and definitions in Section 7.1. We present nonelementary lower bounds for the classes of asynchronous products of BPAs and prefix-recognizable systems in Section 7.2. In Section 7.3, we use results from Section 7.2 to prove nonelementary lower bounds for the compositional method à la Feferman and Vaught for the logics HM and EF over all transition systems, as well as over all finite transition systems.

Bibliographic notes. The results in this chapter have been published in the conference paper [77] (STACS 2012) in joint work with Anthony Widjaja Lin.

[1] On the same note, even μ-calculus over prefix-recognizable systems is only EXP-complete [118, 33].

7.1 Preliminaries

Asynchronous product of systems: Given $k \geq 1$ transition systems $\mathcal{T}_1 = (S_1, \mathsf{A}_1, \{\xrightarrow{a}_1 | \ a \in \mathsf{A}_1\}), \ldots, \mathcal{T}_k = (S_k, \mathsf{A}_k, \{\xrightarrow{a}_k | \ a \in \mathsf{A}_k\})$, where $\mathsf{A}_i \cap \mathsf{A}_j = \emptyset$ for each $i \neq j$, we define its *asynchronous product* $\prod_{i=1}^{k} \mathcal{T}_i = (S, \mathsf{A}, \{\xrightarrow{a} | \ a \in \mathsf{A}\})$, where $S = \prod_{i=1}^{k} S_i$, $\mathsf{A} = \bigcup_{i=1}^{k} \mathsf{A}_i$, and where for each $a \in \mathsf{A}$ we have $(s_1, \ldots, s_k) \xrightarrow{a} (s'_1, \ldots, s'_k)$ if, and only if, $s_i \xrightarrow{a}_i s'_i$ for some $i \in [1, k]$ with $a \in \mathsf{A}_i$ and $s_j = s'_j$ for each $j \in [1, k] \setminus \{i\}$.

Logic: For reasons of simplicity of presentation, we talk about a parametrized variant of the logic EF in this chapter that allows to restrict the set of action labels in the EF operator.

Basic process algebras: We briefly recall basic process algebras. A basic process algebra (BPA) is a tuple $\mathcal{P} = (\Sigma, \mathsf{A}, \Delta)$, where Σ is a finite set of *process constants*, $\mathsf{A} \subseteq \mathsf{Act}$ is a finite set of action labels and Δ is a finite set of *rewrite rules* of the form $u \mapsto_a v$, where $a \in \mathsf{A}$, $u \in \Sigma$ and $v \in \Sigma^*$. The associated transition system $\mathcal{T}(\mathcal{P})$ is defined as $\mathcal{T}(\mathcal{P}) = (\Sigma^*, \mathsf{A}, \{\xrightarrow{a} | \ a \in \mathsf{A}\})$, where $\xrightarrow{a} = \{(uw, vw) \mid u \mapsto_a v \in \Delta, w \in \Sigma^*\}$ for each $a \in \mathsf{A}$. The *size* of the BPA is defined as $|\mathcal{P}| = |\Sigma| + |\mathsf{A}| + \sum_{u \mapsto_a v \in \Delta} (1 + |v|)$.

7.2 Hardness of asynchronous product

We start by proving a nonelementary lower bound for the problem of *model checking* EF *on BPA×BPA*:

MODEL CHECKING EF ON BPA × BPA

INPUT: Two BPAs $\mathcal{P} = (\Sigma, \mathsf{A}, \Delta)$, $\mathcal{P}' = (\Sigma', \mathsf{A}', \Delta')$ with $\mathsf{A} \cap \mathsf{A}' = \emptyset$, a pair of process constants $\langle X, X' \rangle \in \Sigma \times \Sigma'$, and an EF formula φ over $\mathsf{A} \cup \mathsf{A}'$.

QUESTION: Does $(\mathcal{T}(\mathcal{P}) \times \mathcal{T}(\mathcal{P}'), \langle X, X' \rangle) \models \varphi$ hold?

Theorem 7.2 ([77]) *Model checking* EF *on* BPA × BPA *is nonelementary.* □

We then show that this lower bound implies a nonelementary lower bound for model checking HM over the class of asynchronous products of two prefix-recognizable systems.

Proof of Theorem 7.2

The structure of the proof of Theorem 7.2 is as follows. We first show how to encode large counters as EF formulas evaluated over the class of asynchronous products of two BPAs. Such large counters are enforced by the two stacks in the two BPAs, which alternately "guess" an encoding of a counter and "check" the correctness of the encoding. In this chapter we will not provide details how this encoding of large counters can be used to encode computations of Turing machines with a nonelementary membership problem since this is rather standard.

Large counters: The following encoding of large numbers is from [195, 34]. In the following, the notations n and ℓ will range over $\mathbb{N}$. We define the standard *Tower function* $\mathsf{Tower} : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ inductively as $\mathsf{Tower}(0, n) \stackrel{\text{def}}{=} n$ and $\mathsf{Tower}(k, n) \stackrel{\text{def}}{=} 2^{\mathsf{Tower}(k-1,n)}$, for each $k > 0$ and each $n \in \mathbb{N}$.

We define the alphabets $\Omega_\ell \stackrel{\text{def}}{=} \{0_\ell, 1_\ell\}$ and the values $\mathsf{val}(0_\ell) \stackrel{\text{def}}{=} 0$ and $\mathsf{val}(1_\ell) \stackrel{\text{def}}{=} 1$ for each $\ell \geq 0$.

A $(1,n)$-*counter* is a word from Ω_0^n. The *value* $\mathsf{val}(c)$ of some $(1,n)$-counter $c = \sigma_0 \cdots \sigma_{n-1}$ is defined as $\mathsf{val}(c) \stackrel{\text{def}}{=} \sum_{i=0}^{n-1} 2^i \cdot \mathsf{val}(\sigma_i) \in [0, 2^n - 1]$. So the set of values $\mathsf{val}(c)$ for $(1,n)$-counters c equals $[0, 2^n - 1] = [0, \mathsf{Tower}(1,n) - 1]$. An (ℓ,n)-*counter* with $\ell > 1$ is a word $c = c_0\sigma_0 c_1\sigma_1 \ldots c_m\sigma_m$, where $m = \mathsf{Tower}(\ell-1,n) - 1$, each c_i is an $(\ell-1,n)$-counter with $\mathsf{val}(c_i) = i$ and $\sigma_i \in \Omega_{\ell-1}$ for each $i \in [0,m]$. We define $\mathsf{val}(c) \stackrel{\text{def}}{=} \sum_{i=0}^{m} 2^i \cdot \mathsf{val}(\sigma_i)$. Observe that $\mathsf{val}(c) \in [0, \mathsf{Tower}(\ell,n) - 1]$ and the length of each (ℓ,n)-counter is uniquely determined by ℓ and n.

In the following, we define $\Omega'_\ell = \{0'_\ell, 1'_\ell\}$ to be a fresh copy of Ω_ℓ; moreover define $\Sigma_\ell = \bigcup_{i=0}^{\ell} \Omega_i$ and analogously $\Sigma'_\ell = \bigcup_{i=0}^{\ell} \Omega'_i$.

Definition of the two BPAs: For each integer $\ell > 0$, let us define the following simple BPA $\mathcal{P}_\ell = (\Sigma_\ell, \mathsf{A}_\ell, \Delta_\ell)$, where

- $\mathsf{A}_\ell = \Sigma_\ell \cup \overline{\Sigma_\ell}$, where $\overline{\Sigma_\ell} = \{\overline{\sigma} \mid \sigma \in \Sigma_\ell\}$ is a dual copy of Σ_ℓ.
- $\Delta_\ell = \{\tau \mapsto_\sigma \sigma\tau \mid \sigma, \tau \in \Sigma_\ell\} \cup \{\sigma \mapsto_{\overline{\sigma}} \varepsilon \mid \sigma \in \Sigma_\ell\}$.

The transition system $\mathcal{T}(\mathcal{P}_\ell)$ has a fairly regular behavior. The set of states is Σ_ℓ^*. Executing an action $\overline{\sigma} \in \overline{\Sigma_\ell}$ from a state $u \in (\Sigma_\ell)^*$ allows to remove exactly this leftmost symbol σ from u if u is non-empty and begins with σ, otherwise $\overline{\sigma}$ cannot be executed from u. Dually, from every nonempty state $u \in (\Sigma_\ell)^+$ of $\mathcal{T}(\mathcal{P}_\ell)$ we can execute every action $\sigma \in \Sigma_\ell$ yielding the state σu; the only state from which the $\sigma \in \Sigma_\ell$ are not executable is the empty word ε. We define the BPA $\mathcal{P}'_\ell$ analogously to $\mathcal{P}_\ell$ but by priming every symbol. Formally, $\mathcal{P}'_\ell = (\Sigma'_\ell, \mathsf{A}'_\ell, \Delta'_\ell)$, where

- $\mathsf{A}'_\ell = \Sigma'_\ell \cup \overline{\Sigma'_\ell}$, where $\overline{\Sigma'_\ell} = \{\overline{\sigma'} \mid \sigma' \in \Sigma'_\ell\}$ is a dual copy of Σ'_ℓ.
- $\Delta'_\ell = \{\tau' \mapsto_{\sigma'} \sigma'\tau' \mid \sigma', \tau' \in \Sigma'_\ell\} \cup \{\sigma' \mapsto_{\overline{\sigma'}} \varepsilon \mid \sigma' \in \Sigma'_\ell\}$.

Note that the set of states of $\mathcal{T}(\mathcal{P}_\ell) \times \mathcal{T}(\mathcal{P}'_\ell)$ is $(\Sigma_\ell)^* \times (\Sigma'_\ell)^*$. Given a state $s = (u, u') \in (\Sigma_\ell)^* \times (\Sigma'_\ell)^*$, we call u the *left stack of* s and u' the *right stack of* s. So we treat the words u and u' as stacks with their left-most symbols being the top of the stack. Recall that every (ℓ,n)-counter is in particular a word over $\Sigma_{\ell-1}$. We extend this notion to words over $\Sigma'_{\ell-1}$ in the usual way. So each (ℓ,n)-counter will in particular be either a word over $\Sigma_{\ell-1}$ or over $\Sigma'_{\ell-1}$, depending on whether we address the left stack or the right stack. Note that if some word over Σ_k (resp. over Σ'_k) has an (ℓ,n)-counter as a prefix, then the length of this prefix is uniquely determined by ℓ and n.

An *extended* (ℓ,n)-*counter* is either a string $c\sigma$, where either $c \in \Sigma_{\ell-1}^*$ is an (ℓ,n)-counter and $\sigma \in \Omega_\ell$, or a string $c'\sigma'$, where $c' \in (\Sigma'_{\ell-1})^*$ is an (ℓ,n)-counter and $\sigma' \in \Omega'_\ell$.

Next, we define some EF formulas (with primed counterparts for the right stack) for each $\ell, n \in \mathbb{N}$:

1. $\mathsf{count}^{\sigma}_{(\ell,n)}$ for each $\sigma \in \Omega_\ell$ such that $(\mathcal{T}(\mathcal{P}_\ell) \times \mathcal{T}(\mathcal{P}'_\ell), (u,u')) \models \mathsf{count}^{\sigma}_{(\ell,n)}$ if, and only if, for some (ℓ,n)-counter c we have that $c\sigma$ is a prefix of u.

2. $\mathsf{count}^{\sigma'}_{(\ell,n)}$ for each $\sigma' \in \Omega_\ell$ such that $(\mathcal{T}(\mathcal{P}_\ell) \times \mathcal{T}(\mathcal{P}'_\ell), (u, u')) \models \mathsf{count}^{\sigma'}_{(\ell,n)}$ if, and only if, for some (ℓ, n)-counter c' we have that $c'\sigma'$ is a prefix of u'.

3. $\mathsf{xcount}_{(\ell,n)}$ such that $(\mathcal{T}(\mathcal{P}_\ell) \times \mathcal{T}(\mathcal{P}'_\ell), (u, u')) \models \mathsf{xcount}_{(\ell,n)}$ if, and only if, some extended (ℓ, n)-counter $c\sigma$ (for $\sigma \in \Omega_\ell$) is a prefix of u.

4. $\mathsf{xcount}'_{(\ell,n)}$ such that $(\mathcal{T}(\mathcal{P}_\ell) \times \mathcal{T}(\mathcal{P}'_\ell), (u, u')) \models \mathsf{xcount}'_{(\ell,n)}$ if, and only if, some extended (ℓ, n)-counter $c'\sigma'$ (for $\sigma' \in \Omega'_\ell$) is a prefix of u'.

5. $\mathsf{first}_{(\ell,n)}$ (resp. $\mathsf{first}'_{(\ell,n)}$) such that $(\mathcal{T}(\mathcal{P}_\ell) \times \mathcal{T}(\mathcal{P}'_\ell), (u, u')) \models \mathsf{first}_{(\ell,n)}$ (resp. $(\mathcal{T}(\mathcal{P}_\ell) \times \mathcal{T}(\mathcal{P}'_\ell), (u, u')) \models \mathsf{first}'_{(\ell,n)}$) if, and only if, some extended (ℓ, n)-counter $c\sigma$ (resp. $c'\sigma'$) with $\mathsf{val}(c) = 0$ (resp. $\mathsf{val}(c') = 0$) is a prefix of u (resp. u').

6. $\mathsf{last}_{(\ell,n)}$ (resp. $\mathsf{last}'_{(\ell,n)}$) such that $(\mathcal{T}(\mathcal{P}_\ell) \times \mathcal{T}(\mathcal{P}'_\ell), (u, u')) \models \mathsf{last}_{(\ell,n)}$ (resp. $(\mathcal{T}(\mathcal{P}_\ell) \times \mathcal{T}(\mathcal{P}'_\ell), (u, u')) \models \mathsf{last}'_{(\ell,n)}$) if, and only if, some extended (ℓ, n)-counter $c\sigma$ (resp. $c'\sigma'$) with $\mathsf{val}(c) = \mathsf{Tower}(\ell, n) - 1$ (resp. $\mathsf{val}(c') = \mathsf{Tower}(\ell, n) - 1$) is a prefix of u (resp. u').

7. $\mathsf{eq}_{(\ell,n)}$ such that $(\mathcal{T}(\mathcal{P}_\ell) \times \mathcal{T}(\mathcal{P}'_\ell), (u, u')) \models \mathsf{eq}_{(\ell,n)}$ if, and only if, there exist extended (ℓ, n)-counters $c\sigma \in (\Sigma_{\ell-1})^*\Omega_\ell$ and $c'\sigma' \in (\Sigma'_{\ell-1})^*\Omega'_\ell$ such that (i) $c\sigma$ is a prefix of u, (ii) $c'\sigma'$ is a prefix of u', and (iii) $\mathsf{val}(c) = \mathsf{val}(c')$.

8. $\mathsf{inc}_{(\ell,n)}$ (resp. $\mathsf{inc}'_{(\ell,n)}$) such that $(\mathcal{T}(\mathcal{P}_\ell) \times \mathcal{T}(\mathcal{P}'_\ell), (u, u')) \models \mathsf{inc}_{(\ell,n)}$ (resp. $(\mathcal{T}(\mathcal{P}_\ell) \times \mathcal{T}(\mathcal{P}'_\ell), (u, u')) \models \mathsf{inc}'_{(\ell,n)}$) if, and only if, there exist extended (ℓ, n)-counters $c\sigma \in (\Sigma_{\ell-1})^*\Omega_\ell$ and $c'\sigma' \in (\Sigma'_{\ell-1})^*\Omega'_\ell$ such that (i) $c\sigma$ is a prefix of u, (ii) $c'\sigma'$ is a prefix of u', and (iii) $\mathsf{val}(c) + 1 = \mathsf{val}(c')$ (resp. $\mathsf{val}(c') + 1 = \mathsf{val}(c)$).

9. $\mathsf{succ}_{(\ell,n)}$ (resp. $\mathsf{succ}'_{(\ell,n)}$) such that $(\mathcal{T}(\mathcal{P}_\ell) \times \mathcal{T}(\mathcal{P}'_\ell), (u, u')) \models \mathsf{succ}_{(\ell,n)}$ (resp. $(\mathcal{T}(\mathcal{P}_\ell) \times \mathcal{T}(\mathcal{P}'_\ell), (u, u')) \models \mathsf{succ}'_{(\ell,n)}$) if, and only if, there are extended (ℓ, n)-counters $c_1\sigma_1$ and $c_2\sigma_2$ (resp. $c'_1\sigma'_1$ and $c'_2\sigma'_2$) with $\sigma_1, \sigma_2 \in \Omega_\ell$ (resp. $\sigma'_1, \sigma'_2 \in \Omega'_\ell$) such that $c_1\sigma_1c_2\sigma_2$ is a prefix of u and $\mathsf{val}(c_1) + 1 = \mathsf{val}(c_2)$ (resp. $\mathsf{val}(c'_1) + 1 = \mathsf{val}(c'_2)$).

The size of the formulas that we will define will be exponential in ℓ and polynomial in n (both represented in unary). This definition will be given by induction on ℓ. We will start with the following simple observations: $\mathsf{xcount}_{(\ell,n)} \stackrel{\text{def}}{=} \bigvee_{\sigma\in\Omega_\ell} \mathsf{count}^{\sigma}_{(\ell,n)}$, and $\mathsf{xcount}'_{(\ell,n)} \stackrel{\text{def}}{=} \bigvee_{\sigma'\in\Omega'_\ell} \mathsf{count}^{\sigma'}_{(\ell,n)}$. We will now construct several formulas φ that we evaluate on $\mathcal{T}(\mathcal{P}_\ell)\times\mathcal{T}(\mathcal{P}'_\ell)$ expressing properties of the *left stack*. Without making them explicit, we can construct corresponding analogs φ' expressing the respective property on the *right stack*.

Let us proceed by defining the above formulas for the case of $\ell = 1$. We define $\mathsf{count}^{\sigma}_{(1,n)}$ and $\mathsf{count}^{\sigma'}_{(1,n)}$ as follows:

$$\mathsf{count}^{\sigma}_{(1,n)} \stackrel{\text{def}}{=} \langle\overline{\Omega_0}\rangle^n \langle\overline{\sigma}\rangle\mathtt{true} \qquad \text{and} \qquad \mathsf{count}^{\sigma'}_{(1,n)} \stackrel{\text{def}}{=} \left\langle\overline{\Omega'_0}\right\rangle^n \langle\overline{\sigma'}\rangle\mathtt{true}$$

We put $\mathsf{first}_{(1,n)} \stackrel{\text{def}}{=} \langle\overline{0_0}\rangle^n\langle\overline{\Omega_1}\rangle\mathtt{true}$. The definition of $\mathsf{last}_{(1,n)}$ is analogous. We also define $\mathsf{eq}_{(1,n)}$ as $\mathsf{xcount}_{(1,n)} \wedge \mathsf{xcount}'_{(1,n)} \wedge \bigwedge_{i=0}^{n-1}\langle\overline{\Omega_0}\rangle^i\langle\overline{\Omega'_0}\rangle^i \left(\bigwedge_{\sigma\in\Omega_0}(\langle\overline{\sigma}\rangle\mathtt{true} \leftrightarrow \langle\overline{\sigma'}\rangle\mathtt{true})\right)$. The definitions of $\mathsf{inc}_{(1,n)}$ and $\mathsf{succ}_{(1,n)}$ are analogous.

Let us now proceed to the case of $\ell > 1$. We start by defining the formula $\mathsf{count}^{\sigma}_{(\ell,n)}$ for each $\sigma \in \Omega_\ell$. We will achieve this, by making use of the formulas $\mathsf{first}_{(\ell-1,n)}$, $\mathsf{last}_{(j,n)}$ with $j \in [1, \ell-1]$, $\mathsf{xcount}_{(\ell-1,n)}$, and $\mathsf{succ}_{(\ell-1,n)}$. The first two conjuncts of the definition of $\mathsf{count}^{\sigma}_{(\ell,n)}$ are self-explanatory,

$$\mathsf{count}^{\sigma}_{(\ell,n)} \stackrel{\text{def}}{=} \mathsf{first}_{(\ell-1,n)} \wedge \left[\overline{\Sigma_{\ell-1}}^{*}\right] \left(\mathsf{xcount}_{(\ell-1,n)} \rightarrow \left(\mathsf{last}_{(\ell-1,n)} \vee \mathsf{succ}_{(\ell-1,n)} \right) \right) \wedge \mathsf{add}^{\sigma},$$

whereas the formula add^{σ} will express that the symbol σ follows after the top-most (ℓ, n)-counter. Formally we put $\mathsf{add}^{\sigma} \stackrel{\text{def}}{=} \psi^{\sigma}_{\ell-1}$, where

$$\psi^{\sigma}_{j} \stackrel{\text{def}}{=} \begin{cases} \left[\overline{\Sigma_j}^{*}\right] \left(\mathsf{last}_{(j,n)} \rightarrow \psi^{\sigma}_{j-1} \right) & \text{if } j > 1 \\ \left[\overline{\Sigma_1}^{*}\right] \left(\mathsf{last}_{(1,n)} \rightarrow \left\langle \overline{1_0} \right\rangle^{n} \langle \overline{1_1} \rangle \langle \overline{1_2} \rangle \cdots \langle \overline{1_{\ell-2}} \rangle \langle \overline{\Omega_{\ell-1}} \rangle \langle \overline{\sigma} \rangle \mathtt{true} \right) & \text{if } j = 1. \end{cases}$$

Intuitively, the formula $\psi^{\sigma}_{\ell-1}$ jumps to last extended $(1, n)$-counter of the last extended $(2, n)$-counter ... of the last extended $(\ell-1, n)$-counter and expresses that the correct sequence follows from this position. We now define $\mathsf{first}_{(\ell,n)}$ as follows:

$$\mathsf{first}_{(\ell,n)} \stackrel{\text{def}}{=} \mathsf{xcount}_{(\ell,n)} \wedge \left[\overline{\Sigma_{\ell-1}}^{*}\right] \left(\left\langle \overline{\Omega_{\ell-1}} \right\rangle \mathtt{true} \rightarrow \left\langle \overline{0_{\ell-1}} \right\rangle \mathtt{true} \right).$$

The definition of $\mathsf{last}_{(\ell,n)}$ is similar.

We now express $\mathsf{eq}_{(\ell,n)}$, for each $\ell > 1$, as the conjunction of $\mathsf{xcount}_{(\ell,n)} \wedge \mathsf{xcount}'_{(\ell,n)}$ and

$$\left[\overline{\Sigma_{\ell-1}}^{*}\right] \Bigg(\mathsf{xcount}_{(\ell-1,n)} \rightarrow \left(\left\langle \overline{\Sigma'_{\ell-1}}^{*} \right\rangle \left(\mathsf{eq}_{(\ell-1,n)} \wedge \bigwedge_{\sigma \in \Omega_{\ell-1}} \left(\langle \overline{\Sigma^{*}_{\ell-2}} \rangle \langle \overline{\sigma} \rangle \mathtt{true} \leftrightarrow \langle \overline{\Sigma'^{*}_{\ell-2}} \rangle \langle \overline{\sigma'} \rangle \mathtt{true} \right) \right) \right) \Bigg).$$

Let us give some intuition on the formulas $\mathsf{eq}_{(\ell,n)}$ for each $\ell \in [2, k]$: Whenever we pop from the left stack some string from $(\Sigma_{\ell-1})^*$ until on top of the left stack there is some extended $(\ell-1, n)$-counter $c\sigma$, one can remove from the right stack a string from $(\Sigma'_{\ell-1})^*$ yielding an extended $(\ell-1, n)$ counter $c'\tau'$ on top of the right stack such that $\mathsf{val}(c) = \mathsf{val}(c')$ and moreover $\sigma = \tau$ holds.

In analogy to $\mathsf{eq}_{(\ell,n)}$ one can define the formula $\mathsf{inc}_{(\ell,n)}$. Finally, let us define $\mathsf{succ}_{(\ell,n)}$. We put

$$\mathsf{succ}_{(\ell,n)} \stackrel{\text{def}}{=} \left\langle \Sigma'_{\ell} \right\rangle \left\langle (\Sigma'_{\ell-1})^{*} \right\rangle \left(\mathsf{eq}_{(\ell,n)} \wedge \left\langle \overline{\Sigma_{\ell-1}}^{*} \right\rangle \left\langle \overline{\Sigma_{\ell}} \right\rangle \mathsf{inc}'_{(\ell,n)} \right)$$

Intuitively, we the formula $\mathsf{succ}_{(\ell,n)}$ pushes onto the right stack some string that it checks to be a copy of the topmost extended (ℓ, n)-counter of the left stack via $\mathsf{eq}_{(\ell,n)}$, then pops the topmost extended (ℓ, n)-counter of the left stack and then invokes the formula $\mathsf{inc}'_{(\ell,n)}$.

It is easy to see that the formulas given above express the desired properties. Furthermore, we note that the size of each formula is exponential in ℓ and polynomial in n.

By using standard arguments (e.g. see the proof of PSPACE-hardness of EF model checking over pushdown systems in [18]), one can complete the proof of Theorem 7.2 to encode computations of Turing machines. For proving lower bounds on the size of decompositions later, we are rather interested in the word language of stack contents satisfying the above formulas from 1. to 9. For this, we briefly recall the notion of (deterministic) finite automata. A *deterministic finite automaton (*DFA*)* is a tuple $\mathcal{A} = (Q, \Sigma, q_0, \delta, F)$, where Q is a finite set of *states*, Σ is a finite *alphabet*, $q_0 \in Q$ is the *initial state*, $\delta : Q \times \Sigma \to Q$ is the *transition function*, and $F \subseteq Q$ is the set of *final states*. By $L(\mathcal{A}) = \{w \in \Sigma^* \mid \mathcal{A} \text{ accepts } w\}$ we denote the *language* of $\mathcal{A}$. For simplicity we define the *size* of $\mathcal{A}$ is as $|\mathcal{A}| \stackrel{\text{def}}{=} |Q|$.

We will make use of the following lemma in Section 7.3.

Lemma 7.3 ([77]) *Every* DFA *accepting the regular language*

$$L_{\ell,n} \stackrel{def}{=} \{u \in \Sigma_\ell^* \mid \exists u' \in (\Sigma'_\ell)^* : (\mathcal{T}(\mathcal{P}_\ell) \times \mathcal{T}(\mathcal{P}'_\ell), (u, u')) \models \mathsf{xcount}_{(\ell,n)}\}$$

has at least $\mathsf{Tower}(\ell - 1, n) + 1$ *states.* □

PROOF Recall that every extended (ℓ, n)-counter has a length that is uniquely determined by ℓ and n that is at least $\mathsf{Tower}(\ell - 1, n) + 1$. We have

$$L_{\ell,n} = \{c\sigma w \mid w \in \Sigma_\ell^*, c\sigma \text{ is some extended } (\ell, n)\text{-counter}\}.$$

The lemma now follows from the following simple observation: Every DFA $\mathcal{A}$ over some alphabet Σ with $L(\mathcal{A}) = U \cdot \Sigma^*$ for some $\emptyset \subsetneq U \subseteq \Sigma^m$ has at least m states. ■

Lower bounds for HM

We conclude this section by showing how Theorem 7.2 implies a nonelementary lower bound for model checking HM on the asynchronous product of two prefix-recognizable systems. A *prefix-recognizable system* is a tuple $\mathcal{R} = (\Sigma, \mathsf{A}, \Delta)$, where Σ is finite set of process constants, $\mathsf{A} \subseteq \mathsf{Act}$ is a finite set of action labels and Δ is a finite set of rewrite rules of the form $U \mapsto_a V$, where $a \in \mathsf{A}$, and where $U, V \subseteq \Sigma^*$ are regular languages given as DFAs, say. The associated transition system is $\mathcal{T}(\mathcal{R}) = (\Sigma^*, \mathsf{A}, \{\stackrel{a}{\longrightarrow} \mid a \in \mathsf{A}\})$, where $\stackrel{a}{\longrightarrow} = \{(uw, vw) \mid u \in U, v \in V, w \in \Sigma^* \text{ for some rule } U \mapsto_a V \in \Delta\}$ for each $a \in \mathsf{A}$.

One can now construct from a given pair of BPAs $\mathcal{P} = (\Sigma, \mathsf{A}, \Delta)$ and $\mathcal{P}' = (\Sigma', \mathsf{A}', \Delta')$ and a given EF formula φ over $\mathsf{A} \cup \mathsf{A}'$ a pair of prefix-recognizable systems $\mathcal{R} = (\Sigma, \mathsf{A}, \Delta_\mathcal{R})$ and $\mathcal{R}' = (\Sigma', \mathsf{A}', \Delta'_\mathcal{R})$ and some HM formula $\widetilde{\varphi}$ such that $[\![\varphi]\!]_{\mathcal{T}(\mathcal{P})\times\mathcal{T}(\mathcal{P})} = [\![\widetilde{\varphi}]\!]_{\mathcal{T}(\mathcal{R})\times\mathcal{T}(\mathcal{R}')}$ as follows: By [39] one can compute for each $\Gamma \subseteq \mathsf{A}$ (analogously for each $\Gamma' \subseteq \mathsf{A}'$) a pair of regular languages U_Γ and V_Γ (resp. $U_{\Gamma'}$ and $V_{\Gamma'}$) each accepted by DFAs of at most exponential size (in $|\mathcal{P}| + |\mathcal{P}'|$) such that the relation $\stackrel{\Gamma}{\longrightarrow}^*$ over Σ^* (resp. $\stackrel{\Gamma'}{\longrightarrow}^*$ over $(\Sigma')^*$) is exactly $\mathcal{R}(\widetilde{\Gamma}) \stackrel{\text{def}}{=} \{(uw, vw) \mid u \in U_\Gamma, v \in V_\Gamma, w \in \Sigma^*\}$ (resp. $\mathcal{R}'(\widetilde{\Gamma}') \stackrel{\text{def}}{=} \{(uw, vw) \mid u \in U_{\Gamma'}, v \in V_{\Gamma'}, w \in (\Sigma')^*\}$). The latter is even shown for pushdown systems in [39]. Hence we can define

the HM formula $\widetilde{\varphi}$ to emerge from φ by replacing each occurence of $\langle\Gamma^*\rangle$ by $\langle\widetilde{\Gamma}\rangle$ and each occurrence of $\langle(\Gamma')^*\rangle$ by $\langle\widetilde{\Gamma'}\rangle$.

Theorem 7.2 and the previous remark immediately imply the following corollary.

Corollary 7.4 ([77]) *Model checking* HM *on the asynchronous product of two prefix-recognizable systems is nonelementary.* □

We remark that model checking HM on a single prefix-recognizable system is only PSPACE-complete; the upper bound can be shown via reduction to EF model checking pushdown systems, which is in PSPACE by [196].

7.3 Lower bounds for the compositional method for HM and EF

We start by proving nonelementary lower bounds for the Feferman-Vaught type compositional method for HM and EF logics (i.e. Theorem 7.1) already over the class of asynchronous products of *two* transition systems. In Section 7.3 we will then show how our lower bounds can be relativized to the class of all asynchronous products of *two finite* transition systems in the end of this section.

Let us briefly recall decompositions following Theorem 7.1 for EF logic of the asynchronous product of two transition systems. Analogously one can deal with HM. A *decomposition* with respect to the asynchronous product of two transition systems, the first component being defined over action labels A and the second one over A′ (we assume that any two such sets A and A′ are non-empty and disjoint for the rest of this section) is a triple $\mathcal{D} = (\Psi, \Psi', \beta)$, where $\Psi = \{\psi_i\}_{i\in I}$ and $\Psi' = \{\psi'_j\}_{j\in J}$ for index sets I and J, where β is a positive boolean formula with variables ranging over $\{x_i\}_{i\in I} \cup \{x'_j\}_{j\in J}$, each $\psi \in \Psi$ (resp. each $\psi' \in \Psi'$) is an EF formula that is interpreted on the first (resp. second) component, i.e. over A (resp. A′). Recall that such a decomposition has the property that for every pointed transition system $\mathcal{T}$ over A with state s and every pointed transition system $\mathcal{T}'$ over A′ with state s' and every EF formula φ over A ∪ A′ we have

$$(\mathcal{T} \times \mathcal{T}', (s, s')) \models \varphi \quad \Longleftrightarrow \quad \beta[\mu] \text{ is true},$$

where $\mu(x_i) = 1$ if, and only if, $(\mathcal{T}, s) \models \psi_i$ and where $\mu(x'_j) = 1$ if, and only if, $(\mathcal{T}', s') \models \psi'_j$. As expected, the *size* of such a decomposition is defined as $|\mathcal{D}| \stackrel{\text{def}}{=} \sum_{\psi\in\Psi} |\psi| + \sum_{\psi'\in\Psi'} |\psi'| + |\beta|$.

The goal of this section is to prove the following lower bound on the size of decompositions for EF and HM.

Theorem 7.5 ([77]) *The size of decompositions for* EF *(resp.* HM*) formulas in the sense of Theorem 7.1 cannot be bounded by an elementary function. More precisely, there is a family of* EF *(resp.* HM*) formulas* $\{\varphi_\ell \mid \ell \geq 1\}$ *where* φ_ℓ *is defined over some action labels* $\mathsf{A}_\ell \cup \mathsf{A}'_\ell$*, such that* $|\varphi_\ell| \leq \mathsf{exp}(\ell)$*, and such that for every elementary function* $f : \mathbb{N} \to \mathbb{N}$ *there is some* $h \in \mathbb{N}$ *such that every decomposition* $\mathcal{D}$ *for* φ_h *on the class of all asynchronous products of two transition systems over, respectively,* A_h *and* A'_h *satisfies* $|\mathcal{D}| > f(h)$*.* □

Proof of Theorem 7.5

The proof idea for Theorem 7.5 for the case of the logic EF is as follows (we will remark how to adapt it for the logic HM later). We consider the sequence of pairs of BPAs $\{(\mathcal{P}_\ell, \mathcal{P}'_\ell) \mid \ell \geq 1\}$ defined in the previous section, where the set of states of $\mathcal{T}(\mathcal{P}_\ell)$ (resp. $\mathcal{T}(\mathcal{P}'_\ell)$) is Σ_ℓ^* (resp. $(\Sigma'_\ell)^*$). We will show that if a small (i.e. of elementary size) decomposition for EF formulas exists in general, then there is a family of DFAs $\mathcal{A}_\ell$ of size elementary in ℓ with $L(\mathcal{A}_\ell) = L_{\ell,\ell}$ for each ℓ, clearly contradicting Lemma 7.3. To this end, we invoke the result from [18] about the sizes of automata expressing the sets of configurations of BPAs satisfying EF formulas combined with standard constructions from automatic structures.

We first recall the following proposition from [18] about the size of DFAs representing the set of configurations of BPAs satisfying EF formulas.

Proposition 7.6 ([18]) *Given an* EF *formula* φ *and a* BPA $\mathcal{P} = (\Sigma, \mathsf{A}, \Delta)$*, there exists a* DFA $\mathcal{A}_\varphi$ *of size doubly exponential in* $|\mathcal{P}| + |\varphi|$ *with* $L(\mathcal{A}_\varphi) = [\![\varphi]\!]_{\mathcal{T}(\mathcal{P})}$*, i.e.* $\mathcal{A}_\varphi$ *accepts the set of states* u *of* $\mathcal{T}(\mathcal{P})$ *with* $(\mathcal{T}(\mathcal{P}), u) \models \varphi$*.* □

Actually, in [18], the authors construct alternating finite automata with polynomially many states, which can be translated to DFAs of double exponential size (e.g. see [192]).

Define $\{\varphi_\ell \mid \ell \geq 1\}$ as $\varphi_\ell \stackrel{\text{def}}{=} \mathsf{xcount}_{(\ell,\ell)}$ over the action labels A_ℓ and A'_ℓ, where recall that A_ℓ (resp. A'_ℓ) are the action labels of the BPA $\mathcal{P}_\ell$ (resp. $\mathcal{P}'_\ell$) defined in the previous section.

To prove Theorem 7.5, assume to the contrary that the there exist decompositions for EF formulas φ_ℓ, whose sizes can be bounded from above by an elementary function, say by $\mathsf{Tower}(r, |\varphi_\ell|)$ for some *fixed* $r \in \mathbb{N}$. Let $h \in \mathbb{N}$ be a sufficiently large number for the following arguments to work. Let us fix a smallest possible decomposition $\mathcal{D} = (\Psi, \Psi', \beta)$ for the EF formula $\varphi_h = \mathsf{xcount}_{(h,h)}$ over $\mathsf{A}_h \cup \mathsf{A}'_h$. Thus by assumption $|\mathcal{D}| \leq \mathsf{Tower}(r, |\varphi_h|)$. Let $\Psi = \{\psi_i\}_{i \in I}$ and $\Psi' = \{\psi'_j\}_{j \in J}$. Recall that each $\psi_i \in \Psi$ is an EF formula over A_h, and each $\psi'_j \in \Psi'$ is an EF formula over A'_h. Moreover β is a positive boolean formula over the variables $\{x_i\}_{i \in I} \cup \{x'_j\}_{j \in J}$ such that for every state $(u, u') \in (\Sigma_h)^* \times (\Sigma'_h)^*$ of $\mathcal{T}(\mathcal{P}_h) \times \mathcal{T}(\mathcal{P}'_h)$, it is the case that

$$(\mathcal{T}(\mathcal{P}_h) \times \mathcal{T}(\mathcal{P}'_h), (u, u')) \models \varphi_h \quad \iff \quad \beta[\mu] \text{ is true,}$$

where μ is the assignment to β where we have $\mu(x_i) = 1$ if, and only if, $(\mathcal{T}(\mathcal{P}_h), u) \models \psi_i$ and $\mu(x'_j) = 1$ if, and only if, $(\mathcal{T}(\mathcal{P}'_h), u') \models \psi'_j$.

Next, we will use Proposition 7.6 and the small decomposition given by the assumption to construct a DFA for the language $L_{h,h} = \{u \in \Sigma_h^* \mid \exists u \in (\Sigma'_h)^* : (\mathcal{T}(\mathcal{P}_h) \times \mathcal{T}(\mathcal{P}'_h), (u, u')) \models \varphi_h\}$ with less than $\mathsf{Tower}(h-1, h) + 1$ states, which will contradict Lemma 7.3. To do so, we first make the following simple observation that relates the decomposition $\mathcal{D}$ of φ_h and the formula φ_h itself.

Define the EF formula $\widetilde{\beta}$ over $\mathsf{A}_h \cup \mathsf{A}'_h$ to be obtained from the boolean formula β by replacing each variable x_i by ψ_i and each variable x'_j by ψ'_j. Then, since all formulas ψ_i and ψ'_j are also formulas over $\mathsf{A}_h \cup \mathsf{A}'_h$, the EF formula $\widetilde{\beta}$ is also a formula over $\mathsf{A}_h \cup \mathsf{A}'_h$. Moreover, it is easy to see that by assumption we have $[\![\varphi_h]\!]_{\mathcal{T}(\mathcal{P}_h) \times \mathcal{T}(\mathcal{P}'_h)} = [\![\widetilde{\beta}]\!]_{\mathcal{T}(\mathcal{P}_h) \times \mathcal{T}(\mathcal{P}'_h)}$. In fact, the latter

immediately follows from the fact that

$$[\![\psi_i]\!]_{\mathcal{T}(\mathcal{P}_h)\times\mathcal{T}(\mathcal{P}'_h)} = [\![\psi_i]\!]_{\mathcal{T}(\mathcal{P}_h)} \times (\Sigma'_h)^*, \quad \text{and} \tag{7.1}$$

$$[\![\psi'_j]\!]_{\mathcal{T}(\mathcal{P}_h)\times\mathcal{T}(\mathcal{P}'_h)} = \Sigma_h^* \times [\![\psi'_j]\!]_{\mathcal{T}(\mathcal{P}'_h)} \tag{7.2}$$

which can easily be proven by induction on the structure of the formulas ψ_i and ψ'_j since no action labels of $\mathcal{P}_h$ (resp. $\mathcal{P}'_h$) occur in the action labels of ψ'_j (resp. ψ_i). Thus, the goal to obtain a contradiction will be to show that we can find a small DFA for

$$L_1(\widetilde{\beta}) \;=\; \{u \in \Sigma_h^* \mid \exists u' \in (\Sigma'_h)^* : (\mathcal{T}(\mathcal{P}_h) \times \mathcal{T}(\mathcal{P}'_h), (u, u')) \models \widetilde{\beta}\}.$$

Using Proposition 7.6 we obtain DFAs for $[\![\psi_i]\!]_{\mathcal{T}(\mathcal{P}_h)}$ (for each $i \in I$) and $[\![\psi'_j]\!]_{\mathcal{T}(\mathcal{P}'_h)}$ (for each $j \in J$) each of size doubly exponential in, respectively, $|\psi_i| + |\mathcal{P}_h|$ and $|\psi_j| + |\mathcal{P}'_h|$. To obtain a small DFA for $L_1(\widetilde{\beta})$ from these DFAs, we will now perform some simple constructions from automatic structures (e.g. see [184]). We first briefly recall the notion of (binary) automatic relations. Fix a nonempty finite alphabet Σ. A pair of words $(u, w) = (a_1 \cdots a_m, b_1 \cdots b_n) \in \Sigma^* \times \Sigma^*$ can be represented as a word $u \otimes w = c_1 \cdots c_k$ of length $k = \max(m, n)$ in the new alphabet $\Sigma_\perp \times \Sigma_\perp$, where $\Sigma_\perp = \Sigma \cup \{\perp\}$ with a "padding" symbol $\perp \notin \Sigma$, and

$$c_i = \begin{cases} (a_i, b_i) & \text{if } i \leq m, i \leq n \\ (a_i, \perp) & \text{if } i \leq m, i > n \\ (\perp, b_i) & \text{if } i > m, i \leq n. \end{cases}$$

A (binary) relation $R \subseteq \Sigma^* \times \Sigma^*$ is said to be *automatic* if the language $\{u \otimes v \mid (u, v) \in R\} \subseteq (\Sigma_\perp \times \Sigma_\perp)^*$ can be accepted by a DFA (i.e. is regular). We also write $\pi_1(R)$ to be the projection of R to the first component, i.e., $\pi_1(R) = \{u \in \Sigma^* \mid \exists w : (u, w) \in R\}$. The following proposition is folklore (e.g. see [184]):

Proposition 7.7 (folklore) *Given two automatic relations R_1, R_2 accepted by* DFA*s $\mathcal{A}_1$ and $\mathcal{A}_2$, respectively, the following statements hold:*

- *The relation $R_1 \cap R_2$ can be accepted by a* DFA *of size at most $|\mathcal{A}_1| \cdot |\mathcal{A}_2|$.*
- *The relation $R_1 \cup R_2$ can be accepted by a* DFA *of size at most $|\mathcal{A}_1| \cdot |\mathcal{A}_2|$.*
- *The language $\pi_1(R_1) \subseteq \Sigma^*$ can be accepted by a* DFA *of size $2^{O(|\mathcal{A}_1|)}$.* □

Observe now that $[\![\psi_i]\!]_{\mathcal{T}(\mathcal{P}_h)\times\mathcal{T}(\mathcal{P}'_h)}$ (for each $i \in I$) and $[\![\psi'_j]\!]_{\mathcal{T}(\mathcal{P}_h)\times\mathcal{T}(\mathcal{P}'_h)}$ (for each $j \in J$) is an automatic relation over the alphabet $\Sigma = \Sigma_h \cup \Sigma'_h$ that can be accepted by DFAs of size doubly exponential in, respectively, $|\psi_i| + |\mathcal{P}_h| + |\mathcal{P}'_h|$ and $|\psi'_j| + |\mathcal{P}_h| + |\mathcal{P}'_h|$ by Proposition 7.6. The construction of a small DFA $\mathcal{A}$ for the language $L_1(\widetilde{\beta})$ can be done in a bottom-up fashion with respect to $\widetilde{\beta}$ using Proposition 7.7 by firstly taking unions and intersections from the DFAs recognizing $[\![\psi_i]\!]_{\mathcal{T}(\mathcal{P}_h)\times\mathcal{T}(\mathcal{P}'_h)}$ (for each $i \in I$) and $[\![\psi'_j]\!]_{\mathcal{T}(\mathcal{P}_h)\times\mathcal{T}(\mathcal{P}'_h)}$ (for each $j \in J$), and at the end projecting to the first component. All in all, there are constants c_1, c_2 with $c_1 < c_2$ (both independent of h) such that

$$|\mathcal{A}| \;\leq\; \mathsf{Tower}(c_1, |\varphi_h| + |\mathcal{P}_h| + |\mathcal{P}'_h|) \;\leq\; \mathsf{Tower}(c_2, h). \tag{7.3}$$

The latter inequality follows from the fact that $|\mathcal{P}_h| + |\mathcal{P}'_h| \leq \mathsf{poly}(h)$ and $|\varphi_h| \leq \mathsf{exp}(h)$. On the other hand, due to $[\![\varphi_h]\!]_{\mathcal{T}(\mathcal{P}_h)\times\mathcal{T}(\mathcal{P}'_h)} = [\![\widetilde{\beta}]\!]_{\mathcal{T}(\mathcal{P}_h)\times\mathcal{T}(\mathcal{P}'_h)}$ and Lemma 7.3, we must have

$$|\mathcal{A}| \geq \mathsf{Tower}(h-1, h) + 1. \tag{7.4}$$

It is clear that if we choose h sufficiently large, then inequalities (7.3) and (7.4) cannot hold at the same time, a contradiction.

Remark. The proof above can be easily adapted to the case of HM-logic by taking prefix-recognizable systems and the HM formulas of the form $\mathsf{x}\widetilde{\mathsf{count}}_{(\ell,\ell)}$ in analogy to the end of the previous section (instead of BPAs and EF formulas of the form $\mathsf{xcount}_{(\ell,\ell)}$).

Restricting to finite transition systems

Theorem 7.5 gives a nonelementary lower bound for decompositions over the asynchronous product of two general transition systems. This still leaves the possibility that better upper bounds might be possible when we consider only asynchronous products of *finite* transition systems, i.e., the version of Theorem 7.1 when transition systems under consideration are finite. The following theorem shows that this is not the case.

Theorem 7.8 ([77]) *The size of decompositions for* EF *(resp.* HM*) formulas in the sense of Theorem 7.1 cannot be bounded by an elementary function when restricted to the class of finite transition systems.* □

Roughly speaking, this theorem can be proven by combining Theorem 7.5 and the fact that HM and EF logics satisfy "finite model property with respect to a finite set of formulas": a logic $\mathcal{L}$ is said to satisfy the *finite model property with respect to a finite set of formulas* whenever, for every finite set Ξ of $\mathcal{L}$-formulas and every transition system $\mathcal{T}$ with state s there exists a *finite* pointed transition system $\mathcal{T}_\Xi$ with state s_Ξ such that for all $\psi \in \Xi$ we have $(\mathcal{T}, s) \models \psi$ if, and only if, $(\mathcal{T}_\Xi, s_\Xi) \models \psi$.

It simple to check that when restricted to logics that are closed under boolean operations the finite model property with respect to a finite set of formulas is equivalent to the finite model property (for single formulas).

8 Lower bounds for bisimilarity of (higher-order) pushdown systems

In this chapter we present lower bounds for bisimilarity checking on pushdown systems and higher-order pushdown systems. A celebrated result by Sénizergues states that bisimilarity checking of pushdown systems is decidable [167]. Sénizergues' upper bound consists of two semi-decision procedures and unfortunately no complexity-theoretic upper bound is known for this problem to date. The best-known lower bound for bisimilarity of pushdown systems is EXP-hardness by Kučera and Mayr [121] — recently Kiefer proved that EXP-hardness already holds for the subclass basic process algebras [114]. Concerning higher-order pushdown systems it has been open whether bisimilarity is decidable.

Our contributions in this chapter are as follows. In Section 8.1 we provide the detailed construction for undecidability of bisimilarity of order-two pushdown systems. We state further undecidability results on the the *lower-order problem* which asks, roughly speaking, asks to decide if a given order-k pushdown system has a behavior that is inherently bisimilar to an order-k pushdown systems (whether there is no reachable configuration that is bisimilar to an order-k' pushdown system with $k' < k$). In Section 8.2 we a high-level description of our proof that bisimilarity of pushdown systems is nonelementary.

Bibliographic notes. The undecidability results on higher-order pushdown systems have been published in the conference paper [27] (FSTTCS 2012) in joint work with Christopher Broadbent. The nonelementary lower bound on bisimiliarity of pushdown systems has been accepted for publication in the conference paper [10] (LICS 2013) in joint work with Michael Benedikt, Stefan Kiefer and Andrzej Murawski.

8.1 Bisimilarity of order-2 pushdown systems is undecidable

Let us state the main decision problem which we study in this section.

k-PDS-BISIMILARITY
INPUT: A k-PDS $\mathcal{P} = (Q, \mathsf{A}, \Gamma, \Delta)$ and two configurations $q(\alpha), q'(\alpha') \in Q \times \mathsf{Stacks}_k(\Gamma)$.
QUESTION: Does $q(\alpha) \sim q'(\alpha')$ hold in $\mathcal{T}(\mathcal{P})$?

The following proposition is folklore and essentially follows from the fact that every configuration of a k-PDS has only finitely many successors and thus a winning strategy for Attacker can be represented by a finite tree by Kőnig's Lemma, see also [110].

Proposition 8.1 *The problem k-PDS-BISIMILARITY is in Π^0_1 for each $k \geq 1$.* □

Our undecidability proof for k-PDS-BISIMILARITY is a reduction from an appropriate variant of Post's Correspondence problem and inspired from [110]. For two words u, v over some finite alphabet Σ we write $u \preceq v$ if $uw = v$ for some $w \in \Sigma^*$, that is if u is a prefix of v. For a word $w = a_1 \cdots a_n$ with $a_i \in \Sigma$ for each $i \in [1, n]$ we denote its *reverse* by $w^R \stackrel{\text{def}}{=} a_n \cdots a_1$. For a finite (resp. infinite) sequence of finite words $u_1, \ldots, u_n$ (resp. $u_1, u_2, \ldots$) we write $\prod_{i \in [1,n]} u_i \stackrel{\text{def}}{=} u_1 u_2 \cdots u_n$ (resp. $\prod_{i \geq 1} u_i \stackrel{\text{def}}{=} u_1 u_2 \cdots$) to denote their concatentation.

An *instance* of *(Modified) Post's Correspondence Problem* is given by a tuple $\mathcal{X} = (J, \Sigma, h_1, h_2)$, where $J \subseteq [1, n]$ for some $n \geq 1$, Σ is a finite *word alphabet*, and where $h_1, h_2 : J^* \to \Sigma^*$ are homomorphisms. We call $\mathcal{X}$ *increasing* if $|h_1(j)| \leq |h_2(j)|$ for each $j \in J$. We call $\mathcal{X}$ *non-erasing* if $h_1(j), h_2(j) \neq \varepsilon$ for each $j \in J$. A *solution* to $\mathcal{X}$ is a word $w = j_1 \cdots j_\ell \in J^\ell$ with $\ell \geq 1$ and $j_1 = 1$ such that $h_1(w) = h_2(w)$. An *ω-solution* to $\mathcal{X}$ is a mapping $s : \mathbb{N}_+ \to J$ with $s(1) = 1$ such that the following equality over ω-words holds: $\prod_{i \geq 1} h_1(s(i)) = \prod_{i \geq 1} h_2(s(i))$.

Remark 8.2 *When $\mathcal{X}$ is non-erasing and increasing, the following two statements are equivalent for each $s : \mathbb{N}_+ \to J$:*

- *The mapping s is an ω-solution to $\mathcal{X}$.*
- *$s(1) = 1$ and $h_1(s(1) \cdots s(\ell)) \preceq h_2(s(1) \cdots s(\ell))$ for every $\ell \in \mathbb{N}_+$.* □

The classical (finitary) problem MPCP asks, given an instance $\mathcal{X}$, whether $\mathcal{X}$ has a solution. The infinitary variant ω-MPCP asks, given an instance $\mathcal{X}$, whether $\mathcal{X}$ has an ω-solution.

It was shown in [160] that ω-MPCP is Π^0_1-complete. As already observed in [110], Sipser's Σ^0_1-hardness reduction [170] from the halting problem of Turing machines to MPCP can be transferred to a Π^0_1-hardness reduction to ω-MPCP even when restricting instances to be increasing (by only using Steps 1 to 5 and avoiding Steps 6 and 7 in Section 5.2 of [170]). In fact, by inspecting the homomorphisms constructed by Sipser, one can additionally assume that the instances are non-erasing; the latter was not necessary in the undecidability proofs from [110], but is essential in our hardness proofs. This leads us to the following problem.

ω-NONERASING-INCREASING-MPCP

INPUT: An instance $\mathcal{X} = (J, \Sigma, h_1, h_2)$ that is non-erasing and increasing, i.e. such that $h_1(j), h_2(j) \neq \varepsilon$ and $|h_1(j)| \leq |h_2(j)|$ for each $j \in J$.

QUESTION: Does $\mathcal{X}$ have an ω-solution?

The following result is folklore, see [170] for a proof.

Theorem 8.3 ([170]) *The problem ω-NONERASING-INCREASING-MPCP is Π^0_1-complete.* □

We prove Π^0_1-hardness of 2-PDS-BISIMILARITY by giving a many-one reduction from the problem ω-NONERASING-INCREASING-MPCP. For this, let us fix an instance $\mathcal{X} = (J, \Sigma, h_1, h_2)$ of ω-NONERASING-INCREASING-MPCP. We will construct a 2-PDA $\mathcal{P} = (Q, \mathsf{A}, \Gamma, \Delta)$ and two configurations $q([1\bot])$ and $q'([1\bot])$ such that $\mathcal{X}$ has an ω-solution if, and only if, $q([1\bot]) \sim q'([1\bot])$ holds in $\mathcal{T}(\mathcal{P})$.

Overview of the Construction. We start from the pair of configurations $q([1\bot])$ (the initial left configuration) and $q'([1\bot])$ (the initial right configuration), thus both initial configurations consist of just one order-1 stack. We partition the bisimulation game into *three phases*.

Defining $j_1 \stackrel{\text{def}}{=} 1$, in **phase 1** we repeatedly push indices $j_2, j_3, \ldots \in J$ onto the order-1-stack of *both* configurations and we let Defender choose them by using the technique of "Defender's Forcing". The idea is that Defender's job is to push an infinite sequence of indices that is an ω-solution to $\mathcal{X}$ onto both order-1 stacks ad infinitum. At any situation in the game of the form $q([j_\ell \cdots j_1 \bot])$ and $q'([j_\ell \cdots j_1 \bot])$ Attacker can play the action f to challenge Defender by claiming that $h_1(j_1 \cdots j_\ell)$ is not a prefix of $h_2(j_1 \cdots j_\ell)$ in the spirit of Remark 8.2.

This leads us to **phase 2** in which Defender wishes to prove $h_1(j_1 \cdots j_\ell) \preceq h_2(j_1 \cdots j_\ell)$ when the bisimulation game is in the pair $q([j_\ell \cdots j_1 \bot])$ and $q'([j_\ell \cdots j_1 \bot])$. Let $w = j_\ell \cdots j_1 \bot$. From the pair $q([w])$ and $q'([w])$ we let the game get to the pair of configurations $t([w][w][w])$ and $t'([w][w][w])$ by performing two push_2 operations on both configurations. From this position, by again using the "Defender's Forcing" technique and popping on the top-most order-1 stack, we allow Defender to choose a situation of the form $x([u^R j_{k-1} \cdots j_1 \bot][w][w])$ and $x'([u^R j_{k-1} \cdots j_1 \bot][w][w])$, where $1 \leq k \leq \ell$, where u is a prefix (possibly empty) of $h_2(j_k)$, and moreover $h_1(j_1 \cdots j_\ell) = h_2(j_1 \cdots j_{k-1})u$.

From the latter pair of configurations, **phase 3** deterministically prints from the left configuration essentially (plus some intermediate symbols) the string $h_1(j_1 \cdots j_\ell)^R$ by first performing a pop_2, and from the right configuration essentially (plus some intermediate symbols) the string $u^R h_2(j_1 \cdots j_{k-1})^R$ by continuing with the current top order-1-stack.

Since we had three copies of w at the end of phase two, we can now perform a pop_2 followed by a single 'wait' on the left configuration, and two pop_2s on the right configuration, so that both then have stack $[w]$. This allows them both to empty their stacks using the same number of pop_1 operations, allowing our 2-PDS to be *normed*. Thus our suggested definition of normedness does not help recover decidability. Recall that a configuration $p(\alpha)$ of a k-PDS $\mathcal{P}$ is *normed* if there exists some control state q_f in $\mathcal{P}$ with $q_f(\bot_k) \not\xrightarrow{a}$ (emits no a-transition) for each $a \in \mathsf{A}$, and such that every configuration $q(\alpha)$ with $q_0(\alpha_0) \longrightarrow^* q(\alpha)$ we have $q(\alpha) \longrightarrow^* q_f(\bot_k)$.

When describing the rules in detail, we list the rewrite rules of $\mathcal{P}$ in reverse order, i.e. first for **phase 3**, then for **phase 2** and finally for **phase 1**. Adapting the notation from [110], the rewrite rules that are presented in a $\boxed{}$ represent the moves added to implement "Defender's Forcing". These moves allow Defender to render the two configurations equal, and hence trivially bisimilar, if Attacker does not allow Defender to "decide the stack operations".

The Details.

Let $\Gamma \stackrel{\text{def}}{=} J \cup \Sigma$. The set of control states Q, the set of actions A and the transitions Δ of $\mathcal{P}$ are implicitly given by the following rewrite rules. We describe the rules for **phase 3** first. We suggest first reading the lemma that follows after the transitions before reading the transitions themselves.

$x\ \gamma \overset{f}{\hookrightarrow}_{\mathcal{P}} y\ \mathsf{pop}_2$	and	$x'\ \gamma \overset{f}{\hookrightarrow}_{\mathcal{P}} y'\ \mathsf{swap}_\gamma$	for each $\gamma \in \Gamma \cup \{\bot\}$
$y\ a \overset{a}{\hookrightarrow}_{\mathcal{P}} y\ \mathsf{pop}_1$	and	$y'\ a \overset{a}{\hookrightarrow}_{\mathcal{P}} y'\ \mathsf{pop}_1$	for each $a \in \Sigma$
$y\ j \overset{a}{\hookrightarrow}_{\mathcal{P}} y\ \mathsf{swap}_{v^R}$			for each $j \in J$, where $h_1(j) = va$
		$y'\ j \overset{a}{\hookrightarrow}_{\mathcal{P}} y'\ \mathsf{swap}_{v^R}$	for each $j \in J$, where $h_2(j) = va$
$y \bot \overset{\bot}{\hookrightarrow}_{\mathcal{P}} z_1\ \mathsf{swap}_\bot$	and	$y' \bot \overset{\bot}{\hookrightarrow}_{\mathcal{P}} z_1'\ \mathsf{pop}_2$	
$z_1 \bot \overset{p}{\hookrightarrow}_{\mathcal{P}} z\ \mathsf{pop}_2$	and	$z_1'\ j \overset{p}{\hookrightarrow}_{\mathcal{P}} z\ \mathsf{pop}_2$	for each $j \in J$
$z\ j \overset{p}{\hookrightarrow}_{\mathcal{P}} z\ \mathsf{pop}_1$			for each $j \in J$

For the following lemma, observe that from both the initial configurations in the lemma there is a unique maximal (w.r.t. $\preceq$) word that can be traced.

Lemma 8.4 ([27]) *Assume $j_1, \ldots, j_\ell \in J$ with $\ell \geq 1$ and let $0 \leq k \leq \ell$. Assume some 2-stack $\alpha = [u^R j_k \cdots j_1 \bot][j_\ell \cdots j_1 \bot][j_\ell \cdots j_1 \bot]$, where $u \in \Sigma^*$. Then we have*

$$x(\alpha) \sim x'(\alpha) \quad \textit{if, and only if,} \quad h_1(j_1 \cdots j_\ell) = h_2(j_1 \cdots j_k)u.$$

We add the following rules to Δ in order to implement **phase 2**.

$r_1\ j \overset{f}{\hookrightarrow}_{\mathcal{P}} r_2\ \mathsf{push}_2$	and	$r_1'\ j \overset{f}{\hookrightarrow}_{\mathcal{P}} r_2'\ \mathsf{push}_2$	for each $j \in J$
$r_2\ j \overset{f}{\hookrightarrow}_{\mathcal{P}} t\ \mathsf{push}_2$	and	$r_2'\ j \overset{f}{\hookrightarrow}_{\mathcal{P}} t'\ \mathsf{push}_2$	for each $j \in J$
$t \bot \overset{\bot}{\hookrightarrow}_{\mathcal{P}} x\ \mathsf{swap}_\bot$	and	$t' \bot \overset{\bot}{\hookrightarrow}_{\mathcal{P}} x'\ \mathsf{swap}_\bot$	
$\boxed{t\ j \overset{p}{\hookrightarrow}_{\mathcal{P}} t'_\downarrow\ \mathsf{swap}_j}$	and	$t'\ j \overset{p}{\hookrightarrow}_{\mathcal{P}} t'_\downarrow\ \mathsf{swap}_j$	for each $j \in J$
$t\ j \overset{p}{\hookrightarrow}_{\mathcal{P}} \overline{t}\ \mathsf{swap}_j$			for each $j \in J$
$\boxed{t\ j \overset{p}{\hookrightarrow}_{\mathcal{P}} t'_j(w)\ \mathsf{swap}_j}$	and	$t'\ j \overset{p}{\hookrightarrow}_{\mathcal{P}} t'_j(w)\ \mathsf{swap}_j$	for each $j \in J$ and each prefix w of $h_2(j)$
$\overline{t}\ j \overset{\downarrow}{\hookrightarrow}_{\mathcal{P}}\ t\ \mathsf{pop}_1$	and	$t'_\downarrow\ j \overset{\downarrow}{\hookrightarrow}_{\mathcal{P}}\ t'\ \mathsf{pop}_1$	for each $j \in J$
		$\boxed{t'_j(w)\ j \overset{\downarrow}{\hookrightarrow}_{\mathcal{P}} t\ \mathsf{pop}_1}$	for each $j \in J$ and each prefix w of $h_2(j)$
$\overline{t}\ j \overset{\langle j,w\rangle}{\hookrightarrow}_{\mathcal{P}} x\ \mathsf{swap}_{w^R}$	and	$t'_j(w) j \overset{\langle j,w\rangle}{\hookrightarrow}_{\mathcal{P}} x'\ \mathsf{swap}_{w^R}$	for each $j \in J$ and each
	and	$\boxed{t'_\downarrow\ j \overset{\langle j,w\rangle}{\hookrightarrow}_{\mathcal{P}} x\ \mathsf{swap}_{w^R}}$	prefix w of $h_2(j)$
		$\boxed{t'_j(w)\ j \overset{\langle j,v\rangle}{\hookrightarrow}_{\mathcal{P}} x\ \mathsf{swap}_{v^R}}$	for each $j \in J$ and all prefixes v, w of $h_2(j)$ s.t. $v \neq w$

Lemma 8.5 ([27]) *Let $\mu = j_1 \cdots j_\ell \in J^\ell$ with $\ell \geq 1$. Then we have*

$$r_1([\mu^R \bot]) \sim r_1'([\mu^R \bot]) \quad \textit{if, and only if,} \quad h_1(\mu) \preceq h_2(\mu).$$

Finally, let us add the following rules to Δ for implementing **phase 1**.

$\boxed{q\ k \overset{\uparrow}{\hookrightarrow}_{\mathcal{P}} q'_j\ \mathsf{swap}_k}$	and	$q'\ k \overset{\uparrow}{\hookrightarrow}_{\mathcal{P}} q'_j\ \mathsf{swap}_k$	for each $j, k \in J$
$q\ j \overset{\uparrow}{\hookrightarrow}_{\mathcal{P}} \overline{q}\ \mathsf{swap}_j$			for each $j \in J$
$q\ j \overset{f}{\hookrightarrow}_{\mathcal{P}} r_1\ \mathsf{swap}_j$	and	$q' j \overset{f}{\hookrightarrow}_{\mathcal{P}} r'_1\ \mathsf{swap}_j$	for each $j \in J$
$\overline{q}\ k \overset{j}{\hookrightarrow}_{\mathcal{P}} q\ \mathsf{swap}_{jk}$			for each $k, j \in J$
		$q'_j\ k \overset{j}{\hookrightarrow}_{\mathcal{P}} q'\ \mathsf{swap}_{jk}$	for each $k, j \in J$
		$\boxed{q'_j\ k \overset{j'}{\hookrightarrow}_{\mathcal{P}} q\ \mathsf{swap}_{j'k}}$	for each $k, j, j' \in J$ with $j' \neq j$

Lemma 8.6 ([27]) *We have $q([1\bot]) \sim q'([1\bot])$ if, and only if, $\mathcal{X}$ has an ω-solution.* □

It is not hard to verify that both configurations $q([1\bot])$ and $q'([1\bot])$ are normed. For the main result in this section to be stated below, the lower bound follows from Theorem 8.3 and Lemma 8.6, whereas the upper bound is stated in Proposition 8.1.

Theorem 8.7 ([27]) *The problem* 2-PDS-BISIMILARITY *is Π^0_1-complete. The lower bound even holds when both input configurations are normed.* □

The Lower Order Problem

Finally in [27], we studied the lower order problem problem which can be seen as the question whether a given order k-PDS has a reachable configuration that is bisimilar to an order k'-PDS, where $k' < k$. The lower order problem is defined below. In the following, a 0-PDS is simply a finite transition system. It is worth mentioning that the lower order problem is incomparable to the problem of whether an input k-PDS is bisimilar to a k'-PDS; in fact for this latter problem decidability is still open whenever $k \geq 2$. Only very recently decidability of whether a given pushdown system (1-PDS) is bisimilar to a finite system has been announced [104]. Let us define the lower order problem.

LOWER ORDER$_{k,k'}$

INPUT: A k-PDS $\mathcal{P}$ and a configuration $q(\alpha)$ of $\mathcal{P}$.
QUESTION: Does there exist a configuration $r(\beta)$ of $\mathcal{P}$ with $q(\alpha) \longrightarrow^* r(\beta)$ such that $r(\beta) \sim r'(\beta')$, where $r'(\beta')$ is a configuration of some k'-PDS $\mathcal{P}'$?

We have obtained the following additional undecidability result in [27].

Theorem 8.8 ([27]) *The problem* LOWER ORDER$_{k,k'}$ *is Σ^0_1-hard and thus undecidable for each $k \geq 2$ and each $0 \leq k' < k$. The lower bound even holds when the input k-PDS is deterministic.* □

To date, we are only aware of a Σ^0_2 upper bound for the problem LOWER ORDER$_{k,k'}$.

8.2 Bisimilarity of pushdown systems is nonelementary

In this section we sketch a nonelementary lower bound for bisimiliarity of pushdown systems. The latter problem can be defined as follows.

PDS-BISIMILARITY

INPUT: A PDS $\mathcal{P} = (Q, \mathsf{A}, \Gamma, \{\overset{a}{\hookrightarrow} | \ a \in \mathsf{A}\})$ and two configurations $q(w), q'(w') \in Q \times \Gamma^*$.

QUESTION: Does $q(w) \sim q(w')$ hold in $\mathcal{T}(\mathcal{P})$?

We recall that bisimilarity has a natural game-theoretic characterization. Given two transition systems, one can consider a the bisimulation game between the players *Attacker* and *Defender*. They play rounds, in which Attacker fires a transition from one of the systems and Defender has to follow with an identically labeled transition from the other system. In the first round, the chosen transitions must lead from the states to be tested for bisimilarity, while, in each subsequent round, they must start at the states reached after the preceding round. Defender loses if she cannot find a matching transition. In this framework, bisimilarity corresponds to the existence of a winning strategy for Defender.

The game-theoretic reading suggests an intuitive way of reducing halting problems for Turing machines to bisimulation problems, based on constructing bisimulation games that satisfy the following condition: a given Turing machine halts on an input string if, and only if, Defender has a winning strategy. Such games can be viewed as a competition between the players, in which Defender is given an opportunity to exhibit an accepting run and Attacker is equipped with mechanisms to challenge (and verify) the correctness of Defender's construction. The effect of constructing a run by Defender is achieved by allowing Defender to make choices during the game. As the process of playing a bisimulation game naturally favors Attacker as the decision maker, it is not actually clear that the game can be used to express Defender's choice. Nevertheless, it turns out that thanks to the forcing technique of [110], it is possible to construct transition systems in which Defender effectively ends up making choices.

When proving hardness of bisimilarity for classes of computational models, such as pushdown systems, the positions of bisimulation games discussed above must correspond to configurations of the machines. In particular, this means that during the game, players can be thought of as having access to the associated computational resources. For example, in our case, Defender will make moves that store his proposed accepting run on the stack. Additionally, the game can also store some information in the control state of the pushdown system, but since we are interested in finding polynomial-time reductions, these have to be of polynomial size.

Next we give more intuition for our argument by discussing how PSPACE computations can be modeled through bisimulation games, following the argument of Kučera and Mayr [121] (their argument is for EXP, which is equal to alternating PSPACE, but we omit alternation from the discussion, because alternating computation will not be used in our main argument). Let us consider a PSPACE machine $\mathcal{M}$ and an input word. We can code the tape configuration of such a machine by a stack of polynomial size, and we will thus naturally consider a reduction that produces a pair of pushdown systems – in fact, they are the same pushdown systems

but with different control states – whose stack configurations at any point represent sequences of configurations of $\mathcal{M}$ with separators (older configurations occur deeper in the stack). The PDS will have moves that can push new tape symbols of the machine $\mathcal{M}$ on the stacks of each configuration, and we can rely on Defender's Forcing to delegate the choice of such moves to Defender. The control state can be used to make sure that tapes are the of correct size, because each configuration is of polynomial size and we can afford to create polynomially many control states as part of a polynomial-time reduction.

In order to check that Defender's choices amount to a computation history, the pushdown system is able to move into a "verification mode": at this point, suppose the top of the stacks correspond to a cell having position i at time $t+1$; the top stack symbol σ is saved in the control state, the stack is popped until the top element corresponds to cell position i at time t, and then the symbol appearing is compared to σ: if the symbol corresponds to what the transition relation of the machine says it should be, the machines behave in a bisimilar manner, and otherwise they do not. Note that in order to support popping from position i at time $t+1$ to position i at time t, a counter will be required. Because in this case only polynomially many steps are needed, the control state space of the pushdown system can be used for that purpose.

What breaks down in this argument when we try to move to more powerful machines – e.g. EXPSPACE machines? Firstly, tape configurations are now of size 2^n, and hence we can no longer use the control state to verify that the tape configurations are even of the right size. Secondly, the verification of a valid transition can no longer be achieved by having the machines simply pop their stacks in synch with one another – they would not know when they have reached the corresponding cell position at the previous tape configuration.

We deal with the first difficulty by adding *counters* to every cell in the stack content; thus the code of a tape configuration will consist of a sequence of n address bits followed by a tape content. We can use these address bits to know that the end of a tape configuration has occurred, and thus restrict the machines to have separators between configurations. The fact that the addresses really do represent counters moving up sequentially will need to be verified, but for EXPSPACE this can still be done through popping and control states.

The solution to the second difficulty is to perform verification of transitions in a very different way from the PSPACE case. Verification will be carried out only when the machines reach the boundary of a tape configuration. At this point, the machines will first *go out of synch* by one tape configuration – with one machine popping the stack down to the next configuration marker while the other keeps its stack intact. After this, the machines will pop stack symbols, but with one machine emitting symbols corresponding exactly to what it sees, while the other machine emits symbols corresponding to the configuration obtained by applying the transition function to the symbols it sees. Thus, in the second phase, the machines will emit the same symbols *exactly when the two successive configurations obey the transition function*.

The above idea can be extended from EXPSPACE to k-EXPSPACE inductively. Indices that count up to a given tower of exponentials will now precede each tape symbol. The indices used to capture smaller towers will be embedded into those for larger ones. For instance, assuming that $c_0, \cdots, c_{2^n-1}$ are the binary strings representing the numbers $0, \cdots, 2^n - 1$ respectively, the sequence $c_0\sigma_0 \cdots c_{2^n-1}\sigma_{2^n-1}$, where σ_i's are bits, will be used to represent natural numbers from the interval $[0, 2^{2^n} - 1]$. The indexing can be used to enforce that the stack consists of tape configurations of the correct size. The verification that counting indices are incremented

correctly as well as the verification that the tape configurations obey the transition function, can be done using the technique of going out of synch and reading distinct symbols.

Altogether, we get k-EXPSPACE-hardness for all k, and thus a nonelementary lower bound. Our construction can be adapted for normed pushdown systems, i.e. for pushdown systems in which every reachable configuration can reach an empty-stack configuration.

Theorem 8.9 ([10]) *The problem* PDS-BISIMILARITY *is nonelementary. The lower bound even holds when both input configurations are normed, i.e. if every reachable configuration can reach an empty-stack configuration.* □

9 Verifying ground tree rewrite systems

In this chapter we investigate the following verification problems over ground tree rewrite systems GTRS (and the extension RGTRS): (1) model checking against the logic EF and EF_1 (i.e. EF restricted to those formulas in which in the parse tree of the formula, every branch has at most one occurence of the operator EF) and (2) weak and strong bisimilarity checking of GTRS/RGTRS against finite transition systems. These problems are arguably the most basic verification problems over infinite-state systems, especially in the concurrent setting (cf. [139]). Our main contribution is to pinpoint the complexity of these problems.

The starting point of this chapter is that EF model checking over GTRS has a nonelementary complexity, already when considering EF formulas with two occurrences of EF operators that are nested. We remark that a nonelementary lower bound for model checking EF on GTRS is inherited from our lower bound proof for model checking an asynchronous product of two BPA (Theorem 7.2) which was found later than the results presented in this chapter. Nevertheless, we present the proof for this lower bound due to the following reasons: (i) the proof of Theorem 7.2 requires an unbounded nesting of the EF operators, whereas the results in this chapter already provide this lower bound when the input formulas have EF nesting depth at most two, (ii) our lower bound technique allows to prove a nonelementary lower bound for bisimilarity between a given GTRS and a given finite system, and (iii) it turns out that the nesting depth of input EF formulas is a significant source of increase in computational complexity when model checking GTRS. This shows that the existing automata-based algorithms for model checking EF on GTRS (cf. [50, 56, 128]) are in some sense optimal, thus answering a question raised by Löding in [128]. The lower bound proof is achieved by an exponential reduction from the decidable first-order theory over finite words, which is well-known to have a nonelementary complexity [180]. With the same arguments one can also show that Hennessy-Milner logic HM suffices to show the nonelementary lower bound over the more general class of RGTRS.

We then proceed to look at the fragment EF_1 of the EF consisting of formulas with EF operator nesting depth at most one. This fragment is interesting for two reasons. Firstly, as mentioned above, our proof of the nonelementary lower bound for problem (1) over GTRS requires precisely two nested occurrences of EF operators. Secondly, there is a polynomial time reduction from problem (2) to the problem of model checking EF_1 formulas over GTRS if the formulas are represented as DAGs, which are exponentially more succinct than the standard tree representation of formulas. Our result is that the problem of model checking EF_1 (over GTRS) is $\mathsf{P}^{\mathsf{NEXP}}$-complete (i.e. within the second level of the exponential hierarchy). To the best of the author's knowledge this is the only natural problem that is known to be complete this complexity class. The latter result cannot be obtained by simply applying the existing automata-based algorithms for EF-logic model checking (cf. [50, 56, 128]). Moreover, a further analysis of our proof shows that checking bisimilarity between a GTRS and a finite system is solvable in coNEXP. This has substantially decreased the nonelementary complexity gap with the best-

known lower bound for the problem, which is PSPACE. In fact, these proof techniques can be easily applied to derive a better upper bound for verification problems for the class PA, namely bisimilarity checking of PA-processes against finite systems is solvable in coNEXP giving the first elementary upper bound for this problem (cf. [174]).

We then consider two natural extensions of the problem of checking bisimilarity against finite systems over GTRS: We show that bisimilarity against finite systems over the more general class RGTRS is nonelementary. The difficulty of proving a complexity lower bound for bisimilarity checking problem is due to the asymmetry between the power of Attacker and Defender (Attacker is often more powerful) in such games. Known lower bound techniques for bisimilarity checking, a.k.a. "Defender's Forcing"[110], are often implemented by the help of finite control unit, which many infinite-state models have (e.g. pushdown systems and Petri nets). The difficulty of providing Defender's Forcing techniques in the absence of finite control unit is witnessed by the plethora of open problems concerning decidability/complexity of equivalence checking over infinite-state models like PA and PAD processes (cf. [174]). The lack of a global finite control unit often means that Defender does not have an *immediate* way of punishing Attacker (i.e. Forcing him not to do something bad). In the case of GTRS or RGTRS, this means that at any given time Attacker may replace any of (potentially unbounded number of) the subtrees that are present in the current configuration (i.e. a tree). The results presented in this chapter provide the first methods for implementing Defender's Forcing technique over infinite-state models that lack finite-control unit for proving nonelementary lower bounds.

Our results for $(\mathsf{R})\mathsf{GTRS}$ are summarized in Table 9.1.

This chapter is organized as follows. Section 9.1 contains some preliminaries. In Section 9.2, we analyze the complexity of model checking EF and its fragment EF_1 on GTRS and RGTRS. We prove a nonelementary lower bound for bisimilarity checking of RGTRS against finite systems in Section 9.3. Section 9.3 depends on Section 9.2.

Bibliographic notes. The results in this chapter have been published in the conference paper [76] (LICS 2011) in joint work with Anthony Widjaja Lin.

9.1 Preliminaries

The class $\mathsf{P}^{\mathsf{NEXP}}$ denotes deterministic polynomial time with oracle access to NEXP. It is in the second level of exponential hierarchy, which in turn is in EXPSPACE. Unless stated otherwise *reductions* are always polynomial time many-one reductions. We define the tower function in one argument $\mathsf{TOWER} : \mathbb{N} \to \mathbb{N}$ as $\mathsf{TOWER}(0) = 1$ and $\mathsf{TOWER}(n+1) = 2^{\mathsf{TOWER}(n)}$ for each $n \in \mathbb{N}$.

A (binary) *word* is a finite sequence $a_1 a_2 \cdots a_n$, where $a_i \in \{0, 1\}$ for each $i \in [1, n]$ that we also identify with the logical structure $\overline{w} = (U, P_0, P_1, <)$, where $U = [1, n]$ is the universe, unary predicates $P_a = \{i \in [1, n] \mid a_i = a\}$ for each $a \in \{0, 1\}$ and the binary predicate $<$. By a result of Stockmeyer the *first-order theory over (binary) words* is nonelementary [180]. We assume in this chapter that first-order formulas are given in prenex normal form.

We will consider the parametrized variant of the logic EF in this chapter and moreover define, for each $i \geq 0$, the *fragment* EF_i of EF to consist of all those EF formulas φ such that each path

Model checking	GTRS	RGTRS
EF_0	PSPACE-complete	
EF_1	$\mathsf{P}^{\mathsf{NEXP}}$-complete	
$\mathsf{EF}_{k,\ k\geq 2}$	NONELEMENTARY	

Bisimilarity against finite systems	GTRS	RGTRS
$\sim$	PSPACE $\cdots$ coNEXP	
$\approx$	NONELEMENTARY	

Table 9.1: EF_i model checking and bisimilarity against finite systems for GTRS and RGTRS.

in the syntax tree of φ contains at most i occurences of the parametrized EF operator (i.e. of the form $\langle \Gamma^* \rangle$ for some finite set $\Gamma \subseteq \mathsf{Act}$) .

In this chapter we will be interested in the following decision problems.

EF MODEL CHECKING ON (R)GTRS

INPUT: A (R)GTRS $\mathcal{R} = (\Sigma, \mathsf{A}, R)$, a tree $T \in \mathsf{Trees}_\Sigma$ and an EF formula φ.
QUESTION: $(\mathcal{T}(\mathcal{R}), T) \models \varphi$?

The analogous question can be asked for the syntactic fragments EF_i of EF. EF model checking of RGTRS is proven to be decidable in time $\mathsf{TOWER}(O(n))$ in [128]. We also consider bisimilarity checking against finite transition systems.

BISIMILARITY OF (R)GTRS AGAINST FINITE SYSTEMS

INPUT: A (R)GTRS $\mathcal{R} = (\Sigma, \mathsf{A}, R)$, a tree $T \in \mathsf{Trees}_\Sigma$, a finite transition system $\mathcal{T}$ and a state s of $\mathcal{T}$.
QUESTION: Does $T \sim s$ hold?

By results from [107, 122] (see also Theorem 1 and Corollary 1 of [121]) bisimilarity against finite systems can be reduced in polynomial time to model checking EF_1 formulas of the kind $\varphi_1 \wedge [\mathsf{A}^*]\varphi_2$, where φ_1, φ_2 are EF_0 formulas in DAG representation.

9.2 Model Checking EF and its fragments on (regular) ground tree rewrite systems

Our first result is that model checking EF_2 over GTRS has nonelementary complexity, which answers the a question raised by Löding [128].

Theorem 9.1 ([76]) *Model checking* EF_2 *over* GTRS *is nonelementary.* □

This proof of this theorem can easily be adapted to show that model checking EF_0 over RGTRS has nonelementary complexity. This lower bound proof is achieved by an exponential reduction from the decidable first-order theory over finite words, which is well-known to be nonelementary [180]. Roughly speaking, we design our GTRS in such a way that in the first phase it reaches from an input tree a huge tree whose yield (a.k.a. frontier) we interpret as a word, which will correspond to a word that witnesses *satisfiability* of an input first-order formula over finite words. This can be realized by the first occurrence of the EF operator in the input formula. In a second phase we mimic the assignment of variables of the first-order sentence by labeling leaves appropriately. In the third and final phase, we check via a deterministic bottom-up tree automaton whether the huge tree (whose leaves are now labeled with variables of the first-order sentence) satisfies the remaining unquantified subformula. This can be realized by the second occurrence of the EF operator.

Let us now proceed with the proof. Fix a first-order sentence over binary words

$$\psi = \exists x_1 \forall x_2 \ \ldots \ \exists x_{2n-1} \forall x_{2n} \ \varphi(x_1, \ldots, x_{2n}).$$

Without loss of generality we will assume $\overline{w} \not\models \psi$ for each binary word w with $|w| < 2$. Our goal is to compute in exponential time a GTRS $\mathcal{R} = (\Sigma, \mathsf{A}, R)$, some initial tree $\mathsf{start} \in \mathsf{Trees}_\Sigma$, and an EF_2-formula θ such that

$$\exists w \in \{0,1\}^* : \ \overline{w} \models \varphi \quad \text{ if, and only if, } \quad \mathsf{start} \models \theta.$$

We define our set of actions as

$$\mathsf{A} \stackrel{\text{def}}{=} \{a_i \mid i \in [1, 2n]\} \cup \{\mathsf{down}, \mathsf{up}_0, \mathsf{up}_1, \mathsf{up}_2\}$$

and let

$$P \stackrel{\text{def}}{=} (\Big(2^{[1,2n]} \cup \{\bot\}\Big) \times \{0,1\}$$

denote the set of *proper leaf labels*. The first component label $\bot$ will not be relevant in this but in subsequent sections. We define the ranked alphabet $\Sigma = (\Sigma_i)_{i \in \{0,1,2\}}$ of $\mathcal{R}$ as follows, where the set Q will be defined later:

- $\Sigma_0 \stackrel{\text{def}}{=} \{\mathsf{start}\} \cup P \cup Q$,
- $\Sigma_1 \stackrel{\text{def}}{=} \{\mathsf{root}\}$, and
- $\Sigma_2 \stackrel{\text{def}}{=} \{\bigstar\}$.

The regular tree language Combs consists of precisely those trees $T \in \mathsf{Trees}_\Sigma$ such that

(1) $T^{-1}(\mathsf{root}) = \{\varepsilon\}$, i.e. the (one and) only node of T that is labeled with root is the root of T,

(2) for each leaf x of T we have $T(x) \in P$,

(3) for each inner node $x \neq \varepsilon$ of T we have that $T(x) = \bigstar$ and that x has a left child that is a leaf, and

(4) there is at least one inner node x (with $T(x) = \bigstar$) that is the child of ε.

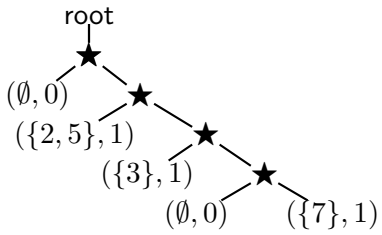
For each $I \subseteq [1, 2n]$ define the regular tree language Combs_I to consist of precisely those combs $T \in \mathsf{Combs}$ such that

(1) for each leaf x of T we have $T(x) \in 2^I \times \{0, 1\}$,

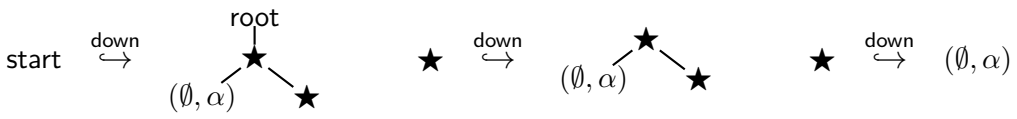
(2) for each two distinct leaves x, x' of T with $T(x) = (J, \alpha)$ and $T(x') = (J', \alpha')$ we have $J \cap J' = \emptyset$, and

(3) $I = \bigcup\{J \mid x \text{ is a leaf of } T \text{ and } T(x) = (J, \alpha)\}$.

Let us give an example for a tree in $\mathsf{Combs}_{\{2,3,5,7\}}$:



Intuitively, think of the sequence of the *second components* of leaf labels of $T \in \mathsf{Combs}_I$ (i.e. the second-component projection of the labels of the yield of T) to correspond to a binary word, and moreover, for each leaf x of T, think of the *first component* of $T(x)$ to correspond to the index set of variables $\{x_1, \ldots, x_n\}$ of φ that have been bound to the corresponding position in the word. Hence every comb in Combs_I corresponds to a unique binary word along with a variable valuation with domain I. By Combs_φ denote the trees from $\mathsf{Combs}_{[1,2n]}$ whose word and variable assignment interpretation satisfies φ.
The following three rewriting rules allow to reach all members of $\mathsf{Combs}_\emptyset$ from the singleton tree start, where $\alpha \in \{0, 1\}$:



Next, we add the following rules that allows to rewrite the leafs of combs (this rewriting will correspond to assigning variables to the leaves), where $\alpha \in \{0,1\}$ and $I \subseteq [1,2n]$:

$$\langle I, \alpha\rangle \quad \overset{a_i}{\hookrightarrow} \quad \langle I \cup \{i\}, \alpha\rangle \qquad \text{for each } i \in [1,2n] \setminus I.$$

In a next step, we compute in exponential time in $|\psi|$ a nondeterministic tree automaton

$$\mathcal{A} = (Q, F, A, \Delta)$$

that accepts Combs_φ, in particular Q is a finite set of *states*, $F \subseteq Q$ is a set of *final states*, Σ is our ranked alphabet from above and $\Delta \subseteq (Q \times Q \times \Sigma_2 \times Q) \cup (Q \times \Sigma_1 \times Q) \cup (\Sigma_0 \times Q)$ is the transition relation. We add the state set Q to A_0 of our GTRS $\mathcal{R}$. Then we add the following rewriting rules to R (which will realize the bottom-up computation of $\mathcal{A}$):

(1) for each rule $(q, q', a, q'') \in \Delta \cap \big(Q^2 \times \Sigma_2 \times Q\big)$ we add the rewriting rule $a(q, q') \overset{\mathsf{up}_2}{\hookrightarrow} q''$,

(2) for each rule $(a, q') \in \Delta \cap (\Sigma_0 \times Q)$ we add the rewriting rule $a \xrightarrow{\mathsf{up}_0} q'$, and

(3) for each rule $(q, \mathsf{root}, q') \in \Delta \cap (Q \times \Sigma_0 \times Q)$ where $q' \in F$ we add the rewriting rule $\mathsf{root}(q') \xrightarrow{\mathsf{up}_1} \mathsf{root}$.

Finally we define θ as

$$\langle \mathsf{down}^* \rangle \langle a_1 \rangle [a_2] \ \cdots \ \langle a_{2n-1} \rangle [a_{2n}] \langle \{\mathsf{up}_0, \mathsf{up}_2\}^* \rangle \langle \mathsf{up}_1 \rangle \, \mathbf{true}.$$

One can easily check that $\exists w \in \{0,1\}^* : \overline{w} \models \psi$ if, and only if, $(\mathcal{T}(\mathcal{R}), \mathsf{start}) \models \theta$, which concludes the proof.

Model checking EF_1 over GTRS is complete for $\mathsf{P}^{\mathsf{NEXP}}$

Our nonelementary lower bound proof above uses nested occurences of two EF operators. Our main result of this section is that prohibitting nested occurences of EF operators yields an elementary model checking complexity.

Theorem 9.2 ([76]) *Over* GTRS *model checking formulas* $\langle \Gamma^* \rangle \varphi$ *with* $\varphi \in \mathsf{EF}_0$ *is in* NEXP. □

Before sketching a proof of this theorem, we mention the following corollary, which can be easily derived by (i) establishing a polynomial space procedure using NEXP oracles (invoked whenever subformulas of the form $\langle \Gamma^* \rangle \psi$ are seen), and (ii) using the fact that $\mathsf{PSPACE}^{\mathsf{NEXP}} = \mathsf{P}^{\mathsf{NEXP}}$ [2].

Corollary 9.3 ([76]) *Model checking* EF_1 *over* GTRS *is in* $\mathsf{P}^{\mathsf{NEXP}}$. □

We now sketch the proof of Theorem 9.2. Let us now suppose that $\langle \Gamma^* \rangle \varphi$ is the given formula, $\mathcal{R} = (\Sigma, \mathsf{A}, R)$ is the given GTRS, and $T_0 \in \mathsf{Trees}_\Sigma$ is the input tree. We wish to check whether $(\mathcal{T}(R), T_0) \models \langle \Gamma^* \rangle \varphi$. Let us compute in polynomial time (cf. [128]) a nondeterministic bottom-up tree automaton $\mathcal{A}$ that recognizes the set $\mathsf{post}^{\Gamma^*}_R(T_0) \overset{\text{def}}{=} \{T \in \mathsf{Trees}_\Sigma \mid T_0 \overset{\Gamma}{\rightarrow}^* T\}$ of trees

of $\mathcal{R}$ reachable from T_0 by applications of rules with labels from Γ. It now suffices to show how to compute nondeterministic bottom-up tree automata that recognize $[\![\varphi]\!]_{\mathcal{T}(\mathcal{R})}$. We do not use the standard automata construction (e.g. [128]) for the set of trees satisfying a given EF_0 formula with respect to a given GTRS since it suffers from a nonelementary blow-up. Given an EF_0 formula, let $\mathsf{mrank}(\varphi)$ be the *modality rank* of φ, i.e., the maximum nesting depth of $\langle\cdot\rangle$ operators in φ.

We will show that $[\![\varphi]\!]_{\mathcal{T}(\mathcal{R})}$ can be expressed as a union of regular tree languages, each of which can be accepted by a nondeterministic bottom-up tree automaton $\mathcal{A}_i$ of exponential size in $|\varphi| + |\mathcal{R}|$. Furthermore, we can check whether some $L(\mathcal{A}_i)$ intersects with $L(\mathcal{A})$ in nondeterministic time exponential in $|\varphi| + |\mathcal{R}|$.

Lemma 9.4 ([76]) *We have* $[\![\varphi]\!]_{\mathcal{T}(\mathcal{R})} = \bigcup_{i\in I} L(\mathcal{A}_i)$*, for a family* $\{\mathcal{A}_i\}_{i\in I}$*, where* $|\mathcal{A}_i| = \mathsf{exp}(|\varphi| + |\mathcal{R}|)$*. One can nondeterministically check whether* $L(\mathcal{A})$ *intersects with some* $L(\mathcal{A}_i)$ *in time* $\mathsf{exp}(|\varphi| + |\mathcal{R}|)$*.*

In fact our proof reveals that the parameter $|\varphi|$ in the above lemma can be replaced by $\mathsf{mrank}(\varphi)$. As a corollary, this yields the same NEXP upper bound for the model checking problem when φ is given as a DAG.

We now give a proof of Lemma 9.4. Let $r = \mathsf{mrank}(\varphi)$. We start by defining a standard equivalence relation on Trees_Σ based on the modality rank of EF_0 formulas: given two trees $T, T' \in \mathsf{Trees}_\Sigma$ and $i \in \mathbb{N}$, write $T \simeq_i T'$ if for every EF_0 formula ψ with $\mathsf{mrank}(\psi) \leq i$ we have $(\mathcal{T}(\mathcal{R}), T) \models \psi$ if, and only if, $(\mathcal{T}(\mathcal{R}), T') \models \psi$. In other words, $T \simeq_i T'$ if, and only if, T and T' satisfy the same formulas of modality rank at most i. It is obvious that $\simeq_i$ is an equivalence relation and that $T \simeq_{i+1} T'$ implies $T \simeq_i T'$. Furthermore, it is well-known that the equivalence relation $\simeq_i$ is of finite index, i.e., the number of equivalence classes of $\simeq_i$ is finite. For each equivalence class $\mathcal{C}$ of $\simeq_r$, it is clear that either $(\mathcal{T}(\mathcal{R}), T) \models \varphi$ for all $T \in \mathcal{C}$, or $(\mathcal{T}(\mathcal{R}), T) \not\models \varphi$ for all $T \in \mathcal{C}$. For the former case, we say that the equivalence class $\mathcal{C}$ is *positive*; otherwise, it is *negative*. Therefore, one idea is to define the family $\{\mathcal{A}_i\}_{i\in I}$ of nondeterministic bottom-up tree automata by associating one such automaton for each positive equivalence class $\mathcal{C}$ of $\simeq_r$. Two problems with this approach arise unfortunately:

- it is not clear how to compute an automaton for each positive equivalence class, and
- this does not reveal an upper bound on the index of $\simeq_r$.

We now define a finer relation $\equiv_i$ (for each $i \in \mathbb{N}$) that will give extra information which will help us solve these two problems. To this end, let K be the maximum number of nodes in the tree appearing in any rewrite rule in $\mathcal{R}$. Also, let $N_i \stackrel{\text{def}}{=} i \cdot K$ for each $i \geq 0$. Given any two trees $T, T' \in \mathsf{Trees}_\Sigma$, we define $T \equiv_i T'$ if for each tree $t \in \mathsf{Trees}_\Sigma$ with at most N_i nodes at least one of the following is true:

- the number of times t appears as a subtree of T equals the number of times t appears as a subtree of T',
- the number of times t appears as a subtree exceeds N_i both for T and T'.

In other words, $T \equiv_i T'$ if, and only if, each subtree with at most N_i nodes appears in T and T' the same number of times up the threshold N_i. In the following, we let $\mathsf{Trees}_\Sigma^{\leq N_i}$ denote the set of trees from Trees_Σ with at most N_i nodes.

As before, it is easy to check that $\equiv_i$ is an equivalence relation and that $T \equiv_{i+1} T'$ implies $T \equiv_i T'$. To complete the proof of Lemma 9.4, one can proceed as follows:

(1) show that $\equiv_i$ is finer than $\simeq_i$,

(2) checking whether a function $f : \mathsf{Trees}_\Sigma^{\leq N_i} \to [0, N_i]$, representing the number of occurrences of subtrees with at most N_i nodes, actually describes an equivalence class of $\equiv_i$ can be done rather efficiently,

(3) testing whether an equivalence class of $\equiv_i$ is positive (with respect to φ) can be done rather efficiently, and

(4) for each positive equivalence class $\mathcal{C}$ of $\equiv_i$, a nondeterministic bottom-up tree automaton $\mathcal{A}_\mathcal{C}$ recognizing $\mathcal{C}$ can be computed rather efficiently.

As we will see, these will imply Lemma 9.4. For step (1) the following lemma can be shown.

Lemma 9.5 ([76]) *For any two trees $T, T' \in \mathsf{Trees}_\Sigma$, we have that $T \equiv_i T'$ implies $T \simeq_i T'$.* □

Intuitively, this lemma holds since satisfaction of EF_0 formulas of modality rank i is only affected by the number of occurences of trees of depth N_i (up to some threshold).

Let us now proceed to step (2). Recall that each equivalence class $\mathcal{C}$ of $\equiv_r$ can be described by a function from $f_\mathcal{C} : \mathsf{Trees}_\Sigma^{\leq N_r} \to [0, N_r]$. The converse, however, is false, e.g., it is impossible to have a class $\mathcal{C}$ with $f_\mathcal{C}(T) > 1$ for a tree T with two nodes but $f_\mathcal{C}(T') = 0$ for all trees T' with one node. Also note that the special case where $f(T) = 0$ for all $T \in \mathsf{Trees}_\Sigma^{\leq N_r}$ is impossible for an equivalence class since trees have nonempty domain by definition. Therefore, we need to be able to check whether a given function $f : \mathsf{Trees}_\Sigma^{\leq N_r} \to [0, N_r]$ *actually* describes an equivalence class in $\equiv_r$. To this end, recall first that any function f that describes an equivalence class of $\equiv_r$ counts each subtree of trees T in $\mathsf{Trees}_\Sigma^{\leq N_r}$ with $f_\mathcal{C}(T) > 0$, i.e., if t is a subtree of T, then t contributes to the value of $f(t)$. We will first define a new function $g : \mathsf{Trees}_\Sigma^{\leq N_r} \to [0, N_r]$ that avoids "double counting". This can be done by the following algorithm:

- Set $g(T) := 0$ for all $T \in \mathsf{Trees}_\Sigma^{\leq N_r}$ and repeat the following for each $T \in \mathsf{Trees}_\Sigma^{\leq N_r}$ with $f(T) > 0$ (ordered by the number of nodes, starting from the largest):
  (1) Let $f(T) := f(T) - 1$,
  (2) $g(T) := g(T) + 1$,
  (3) Go through all nodes u of T (except when u is the root of T) and substract $f(T^{\downarrow u})$ by 1 (if becomes negative, then terminate abruptly).

Observe that if this algorithm terminates abruptly, then f does not actually describe an equivalence class of $\mathcal{C}$. Furthermore, the algorithm runs in time exponential in $r + |\mathcal{R}|$ (recall that $r \leq |\varphi|$) simply because $|\mathsf{Trees}_\Sigma^{\leq N_r}| \leq \mathsf{exp}(r + |\mathcal{R}|)$ can be shown. Now, suppose that the function g has been successfully computed from the given function f. This implies that g describes

a *forest* F with each tree $T \in \mathsf{Trees}_\Sigma^{\leq N_r}$ occurring $g(T)$ many times. The original function f then describes an equivalence class if, and only if, such a forest can be further "connected into a big tree". This last check can be done using the following lemma.

Lemma 9.6 ([76]) *The function $f : \mathsf{Trees}_\Sigma^{\leq N_r} \to [0, N_r]$ describes an equivalence class in $\equiv_r$ if, and only if, the function $g : \mathsf{Trees}_\Sigma^{\leq N_r} \to [0, N_r]$ (and the forest F corresponding to it) can be successfully computed from f by the above algorithm and that one of the following conditions are satisfied:*

(1) $\sum_{T \in \mathsf{Trees}_\Sigma^{\leq N_r}} g(T) = 1$.

(2) $\sum_{T \in \mathsf{Trees}_\Sigma^{\leq N_r}} g(T) > 1$, *and for some letter a with some rank $h \in \mathbb{N}$ and some trees $T_1, \ldots, T_h$ occuring in the forest F, the tree $a(T_1, \ldots, T_h)$ has more than N_r nodes.* □

Observe that this lemma completes step (2) since this test can be performed in time exponential in $r + |\mathcal{R}|$.

We now proceed to step (3). This step is rather easy since checking whether an equivalence class $\mathcal{C}$ of $\equiv_r$ described by a function $f_\mathcal{C} : \mathsf{Trees}_\Sigma^{\leq N_r} \to [0, N_r]$ is positive can be done in time exponential in $r + |\mathcal{R}|$. Intuitively, the idea is to pick a representative T of $\mathcal{C}$ of exponential size and compute a finite transition system consisting of the neighborhood of T up to depth r. It turns out that the finite transition system also has size exponential in $r + |\mathcal{R}|$. Therefore, we may use the standard linear-time algorithm for model checking HM (i.e. EF_0) formulas over finite transition systems.

We now proceed to step (4), which is the final step. For this, we need to show how to compute a nondeterministic bottom-up tree automaton recognizing an equivalence class $\mathcal{C}$ of $\equiv_r$ described by a function $f_\mathcal{C} : \mathsf{Trees}_\Sigma^{\leq N_r} \to [0, N_r]$.

Lemma 9.7 ([76]) *Given a function $f : \mathsf{Trees}_\Sigma^{\leq N_r} \to [0, N_r]$ that witnesses an equivalence class $\mathcal{C}$ of $\equiv_r$, we can compute a nondeterministic bottom-up tree automaton recognizing precisely $\mathcal{C}$ in time $|f|^{\mathsf{poly}(r+|\mathcal{R}|)} \cdot \mathsf{exp}(r + |\mathcal{R}|)$.* □

This lemma can be proven as follows. First, compute the function $g : \mathsf{Trees}_\Sigma^{\leq N_r} \to [0, N_r]$ using the above algorithm, which avoids double counting of subtrees. Let U denote all trees $t \in \mathsf{Trees}_\Sigma^{\leq N_r}$ such that $g(t) = N_r$. Let $t_1, \ldots, t_m$ be an enumeration of all trees $t \in \mathsf{Trees}_\Sigma^{\leq N_r}$ with $g(t) > 0$ *without* counting multiplicities.

One can now design a nondeterministic bottom-up tree automaton that counts that precisely $g(t_i)$ many nodes v occur such that the subtree rooted at v equals t_i, and that an arbitrary number of nodes v can occur such that the subtree rooted at v is a tree in U. It is easy to see that such an automaton of size exponential in r and $|\mathcal{R}|$ can be computed.

To summarize, the proof of Lemma 9.4 can now be done as follows. The nondeterministic bottom-up tree automata $\mathcal{A}_i$ in the statement of Lemma 9.4 will correspond to positive equivalence classes $\mathcal{C}$ described by some functions $f_\mathcal{C} : \mathsf{Trees}_\Sigma^{\leq N_r} \to [0, N_r]$. Using the last step above, the automaton $\mathcal{A}_i$ can be computed in time exponential in $r + |\mathcal{R}|$ if f is given as an input. Checking whether $L(\mathcal{A}) \cap L(\mathcal{A}_i) \neq \emptyset$ for some i requires us to nondeterministically guess

one such function f, check whether it describes an equivalence class, compute the automaton $\mathcal{A}_i$ corresponding to it, and check for language intersection with $\mathcal{A}$ in the standard way.

Let us discuss the ideas of a matching lower bound for EF_1.

Lemma 9.8 ([76]) *Over* GTRS *model checking formulas* $\langle \mathsf{A}^* \rangle \varphi$*, where* $\varphi \in \mathsf{EF}_0$*, is* NEXP*-hard.* □

PROOF (SKETCH) The reduction is from the $2^n \times 2^n$ tiling problem [151]. The idea is to reach via $\xrightarrow{\mathsf{A}}{}^*$ some binary tree with superleafs, where a *superleaf* is a tree of depth one whose root has arity $2n$. Each child of a superleaf will either have a nullary symbol b_0 or b_1, where the root of a superleaf contains a tile type. Each superleaf corresponds to a grid element $(i, j) \in [0, 2^n - 1] \times [0, 2^n - 1]$, where the nullary symbols of the first n (resp. last n) children encode i (resp. j) in binary. The formula φ is now a conjunction of EF_0 formulas expressing the following:

(1) a superleaf for $(0, 0)$ exists,

(2) whenever there are two superleafs corresponding to the same (i, j), then their tile types are the same,

(3) if there is a superleaf for (i, j) with $i < 2^n - 1$ (resp. $j < 2^n - 1$), then there is a superleaf for $(i + 1, j)$ (resp. $(i, j + 1)$), and finally

(4) the horizontal and vertical tiling conditions hold for every superleaf. ■

By encoding *circuit value* into nodes of trees (gates and its evaluations will be represented in nodes in the tree) and invoking a subroutine to the trees that simulate the $2^n \times 2^n$ tiling problem one can prove a matching lower bound for EF_1.

Theorem 9.9 ([76]) *Over* GTRS *model checking* EF_1 *is hard for* $\mathsf{P}^{\mathsf{NEXP}}$. □

9.3 Bisimilarity checking of (regular) ground tree rewrite systems against finite systems

Since bisimilarity checking against finite systems can be reduced to model checking EF_1 formulas in DAG representation, the following theorem is known.

Theorem 9.10 ([128, 107]) *Bisimilarity checking of* RGTRS *against finite systems is decidable in time* $\mathsf{TOWER}(O(n))$. □

Over GTRS, however, we obtain an elementary upper bound. It can be derived via a reduction to model checking formulas of the kind $\varphi_1 \wedge [\mathsf{A}^*]\varphi_2$, where φ_1 and φ_2 are EF_0 DAG-formulas and then applying our upper bound result from Theorem 9.2. This technique can also be used to prove a coNEXP upper bound for bisimilarity checking of PA-processes against finite systems (cf. [174]).

Theorem 9.11 ([76]) *Bisimilarity checking of* GTRS *against finite systems is in* coNEXP. □

As a main result of this section we prove that bisimilarity between a regular ground tree rewrite system and a finite system has nonelementary complexity.

Theorem 9.12 ([76]) *Bisimilarity checking of* RGTRS *against finite systems is nonelementary.*

Since from every RGTRS $\mathcal{R}$ one can compute in polynomial time a GTRS that is weakly bisimilar to $\mathcal{R}$ by Theorem 3.3 we obtain the following corollary. The author and Anthony Widjaja Lin were not aware of Theorem 3.3 at the time when the paper [76] was finished and hence gave a self-contained proof of the following corollary in [76].

Corollary 9.13 ([75, 76]) *Weak bisimilarity checking of* GTRS *against finite systems is nonelementary.* □

Although the proof of Theorem 9.12 also goes via an exponential reduction from the first-order theory over finite words, due to the lack of finite control unit in (R)GTRS it is not at all merely an adaptation of the proof of the nonelementary lower bound for EF_2 model checking over GTRS from the previous section. Roughly speaking, we implement Defender's Forcing technique by providing rewriting rules of the form $L \hookrightarrow T$, where L is a regular tree language and T is an explicit ranked tree. Such rules will allow Defender to punish Attacker in case he did not play in a way that corresponds to evaluating the first-order sentence on the huge tree. However, the biggest obstacle we have to overcome is the possibility of Attacker rewriting the leaves of the tree (that corresponds to the word where the first-sentence is evaluated) in a chaotic way since the leaves cannot communicate with each other.

Let us proceed to the proof of Theorem 9.12. We reuse some of the notation that was introduced in Section 9.2. Again, let us fix a first-order sentence interpreted over binary words

$$\psi = \exists x_1 \forall x_2 \ \ldots \ \exists x_{2n-1} \forall x_{2n} \varphi(x_1, \ldots, \varphi_{2n})$$

and let us assume again $\overline{w} \not\models \psi$ for each binary word w with $|w| < 2$. Our goal is to compute in exponential time an RGTRS $\mathcal{R} = (\Sigma, \mathsf{A}, R)$, some initial tree $\mathsf{start} \in \mathsf{Trees}_\Sigma$, some finite transition system $\mathcal{T} = (C, \mathsf{A}, \{\xrightarrow{a}_{\mathcal{T}} | \ a \in \mathsf{A}\})$, and a configuration $s_\emptyset \in C$ such that

$$\exists w \in \{0,1\}^* : \overline{w} \models \psi \qquad \text{if, and only if,} \qquad \mathsf{start} \not\sim s_\emptyset.$$

We call a subset $I \subseteq [1, 2n]$ *game-conform* if $I = [1, k]$ for some $k \in [0, 2n]$ and *non-game-conform* otherwise. Analogously, we call a comb $T \in \mathsf{Combs}_I$ *game-conform* (resp. *non-game-conform*) if I is game-conform (resp. non-game-conform). Each game-conform comb $T \in \mathsf{Combs}_{[1,k]}$ naturally induces a valuation ν_T of variables with indices from $[1, k]$ to positions of the yield string defined by T. Let $\varphi[\nu_T]$ denote the formula that is obtained from φ by replacing the information given by ν_T and which is evaluated on T in the expected way. This can be extended to define $\psi[\nu_T]$. Hence, e.g. $\psi[\nu_T]$ is of the form $\exists x_{k+1} \cdots \forall x_{2n} \varphi[\nu_T]$ in case k is even.

In case $I \subset [1, 2n]$ and $i \in [1, 2n] \setminus I$ we say a tree $T' \in \mathsf{Combs}_{I \cup \{i\}}$ is an *i-extension* of T if T' can be obtained from T by choosing exactly one leaf x and replacing its label $T(x) = (J, \alpha)$ by $(J \cup \{i\}, \alpha)$. Recall that by Combs_φ we denote the trees from $\mathsf{Combs}_{[1,2n]}$ whose word and variable assignment interpretation satisfies φ. Likewise let $\mathsf{Combs}_{\overline{\varphi}}$ denote the trees from $\mathsf{Combs}_{[1,2n]}$ whose word and variable assignment interpretation does not satisfy φ.

We define the regular tree language $\mathsf{Combs}_\perp = \mathsf{Combs} \setminus \bigcup_{I \subseteq [1,2n]} \mathsf{Combs}_I$. In other words, $\mathsf{Combs}_\perp$ consists of those combs $T \in \mathsf{Combs}$ that satisfy

(1) there is some leaf x of T with $T(x) \in \{\perp\} \times \{0,1\}$ *or*

(2) there are two distinct leaves x, x' of T with $T(x) = (J, \alpha)$ and $T(x') = (J', \alpha')$ such that $J \cap J' \neq \emptyset$.

For each game-conform I we will introduce *two* configurations s_I and $\overline{s_I}$ in $\mathcal{T}$. For each non-game-conform I we have *one* corresponding configuration u_I in $\mathcal{T}$. In addition our finite system $\mathcal{T}$ has the configurations succ and fail. We define as action labels $\mathsf{A} \stackrel{\text{def}}{=} \{a_i \mid i \in [1,2n]\} \cup \{\varphi\}$.

The idea of the bisimulation game and difficulties

We remark that we have not yet defined the RGTRS $\mathcal{R}$ nor the finite transition system $\mathcal{T}$. Yet, the high level idea of the bisimulation game goes as follows, and uses Defender's Forcing techniques as e.g. in [121]:

- (**initial round**) Attacker chooses from our initial tree start a comb T from $\mathsf{Combs}_{[1,1]}$ by moving along $\mathsf{start} \xrightarrow{a_1} T$ for which he claims that $T \models \psi[\nu_T]$ holds. Defender will only be able to respond $s_\emptyset \xrightarrow{a_1}_{\mathcal{T}} s_{[1,1]}$ in $\mathcal{T}$. Hence the new pebble configuration is $(T, s_{[1,1]})$.
- Next, we repeat the following round game, where the current pebble configuration is $(T, s_{[1,k]})$ where $T \in \mathsf{Combs}_{[1,k]}$ for each round $k = 1, \ldots, 2n-1$:
  - (**universal round**) If k is odd, then Attacker is supposed to move in $\mathcal{T}$, namely $s_{[1,k]} \xrightarrow{a_{k+1}}_{\mathcal{T}} s_{[1,k+1]}$ although the move $s_{[1,k]} \xrightarrow{a_{k+1}}_{\mathcal{T}} \overline{s_{[1,k+1]}}$ is possible. Defender is now forced to move in $\mathcal{T}(\mathcal{R})$, namely $T \xrightarrow{a_{k+1}} T'$ for some $k+1$-extension T' of T. This response corresponds to the universal quantification $\forall x_{k+1}$ in ψ.
  - (**existential round**) If k is even, then Attacker is supposed to move in $\mathcal{T}(\mathcal{R})$, namely $T \xrightarrow{a_{k+1}} T'$ for some $k+1$-extension T' of T. This move corresponds to the existential quantification $\exists x_{k+1}$ in ψ. Defender's only possible response in $\mathcal{T}$ is $s_{[1,k]} \xrightarrow{a_{k+1}}_{\mathcal{T}} s_{[1,k+1]}$.
- (**final round**) Finally, when we are in the pebble configuration $(T, s_{[1,2n]})$, where $T \in \mathsf{Combs}_{[1,2n]}$, the action φ can be performed that allows Attacker to win (via a rule in $\mathcal{R}$ that contains Combs_φ on the left-hand side) if, and only if, $T \models \varphi[\nu_T]$.

In order to implement such a game, several difficulties arise. Let us discuss these difficulties for the universal round (the existential round can be treated dually) and give solutions to them. The question is: In the universal round, how can we force Attacker to make in $\mathcal{T}$ the move $s_{[1,k]} \xrightarrow{a_{k+1}}_{\mathcal{T}} s_{[1,k+1]}$?

- **1. Difficulty:** What if Attacker moves $s_{[1,k]} \xrightarrow{a_{k+1}}_{\mathcal{T}} \overline{s_{[1,k+1]}}$ (which will exist in $\mathcal{T}$)? **Solution:** We add the rule $\mathsf{Combs}_{[1,k]} \overset{a_{k+1}}{\hookrightarrow} \overline{s_{[1,k+1]}}$ to $\mathcal{R}$ such that Defender has the possibility to threaten syntactic equivalence by responding $T \xrightarrow{a_{k+1}} \overline{s_{[1,k+1]}}$ in $\mathcal{T}(\mathcal{R})$ and hence wins.

- **2. Difficulty:** What if Attacker moves $T \xrightarrow{a_{k+1}} T'$ in $\mathcal{T}(\mathcal{R})$ for some $k+1$-extension of T rather than playing in $\mathcal{T}$? **Solution:** Defender can react in $\mathcal{T}$, depending on whether $T' \models \psi[\nu_{T'}]$ or $T' \not\models \psi[\nu_{T'}]$. In case $T' \models \psi[\nu_{T'}]$ she can move to $\overline{s_{[1,k+1]}}$ and can win. In case $T' \not\models \psi[\nu_{T'}]$ she can move to $s_{[1,k+1]}$ and can win.

- **3. Difficulty:** What if Attacker plays $T \xrightarrow{a_i} T'$ where $i \in [1,k]$? I.e. Attacker plays an action that has already been played and thus clearly $T' \in \mathsf{Combs}_\perp$. **Solution:** We allow a simple transition to a configuration in $\mathcal{T}$ from which Defender can surely win since surely $T' \in \mathsf{Combs}_\perp$ and hence $T' \notin \mathsf{Combs}_\varphi$.

- **4. Difficulty:** What if Attacker plays in $T \xrightarrow{a_i} T'$ in $\mathcal{T}(\mathcal{R})$ where $i > k+1$? I.e. Attacker deviates from playing a sequence of actions $a_1 \cdots a_k$ that correspond to assigning variables to positions of the yield string of the tree. We also say that the current pebble configuration is non-game-conform. **Solution:** We allow in $\mathcal{T}$ a special transition for Defender $s_{[1,k]} \xrightarrow{a_i} u_{[1,k]\cup\{i\}}$ that allows her to win.

The solutions to Difficulty 1 and 2 are standard and are similar to a technique elaborated in [121]. The solution to Difficulty 3 is straightforward. The real difficulty *in the absence of a finite control* (e.g. pushdown systems have a finite control) in the game is Difficulty 4. We have to provide configurations in $\mathcal{T}$ that allow to remember the set of variables in the current tree T that have been assigned. The difficulty that now arises is that Attacker can continue labeling leafs in T and pretend some moves later that the current tree T is game-conform all of a sudden (and hence threaten to play the above-mentioned punishing moves for instance). We have to carefully design transitions in $\mathcal{T}$ that sooner or later punish Attacker since he was the one who deviated from playing game-conform.

The finite system:

We now define the outgoing transitions of $s_{[1,k]}$ and of $\overline{s_{[1,k]}}$, *for each possible* $k \in [0, 2n-1]$:

(1) $s_{[1,k]} \xrightarrow{a_{k+1}}_{\mathcal{T}} s_{[1,k+1]}$,

(2) $\overline{s_{[1,k]}} \xrightarrow{a_{k+1}}_{\mathcal{T}} \overline{s_{[1,k+1]}}$,

(3) $s_{[1,k]} \xrightarrow{a_{k+1}}_{\mathcal{T}} \overline{s_{[1,k+1]}}$ if k is odd,

(4) $\overline{s_{[1,k]}} \xrightarrow{a_{k+1}}_{\mathcal{T}} s_{[1,k+1]}$ if k is even,

(5) $s_{[1,k]} \xrightarrow{a_i}_{\mathcal{T}} s_{[1,2n]}$ for each $i \in [1,k]$,

(6) $\overline{s_{[1,k]}} \xrightarrow{a_i}_{\mathcal{T}} s_{[1,2n]}$ for each $i \in [1,k]$,

(7) $s_{[1,k]} \xrightarrow{a_i}_{\mathcal{T}} u_{[1,k+1]\cup\{i\}}$ for each $i \in [k+2, 2n]$,

(8) $\overline{s_{[1,k]}} \xrightarrow{a_i}_{\mathcal{T}} u_{[1,k+1]\cup\{i\}}$ for each $i \in [k+2, 2n]$,

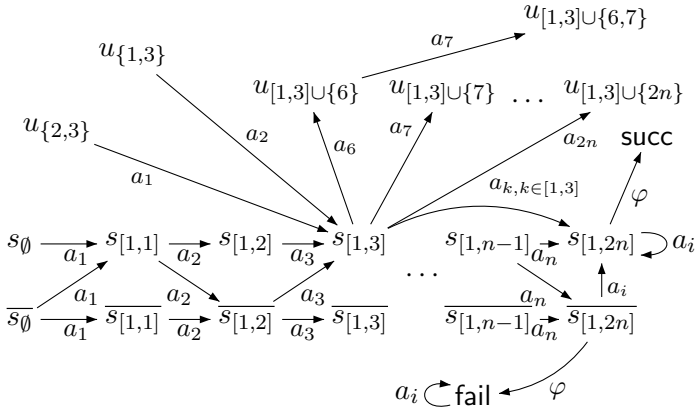


Figure 9.1: A snapshot of $\mathcal{T}$ and the outgoing transitions of $s_{[1,3]}$ in the strong bisimulation game (i ranges over $[1, 2n]$).

(9) $s_{[1,2n]} \xrightarrow{\varphi}_{\mathcal{T}} \mathsf{succ}$,

(10) $\overline{s_{[1,2n]}} \xrightarrow{a_i}_{\mathcal{T}} s_{[1,2n]}$ for each $i \in [1, 2n]$,

(11) $\overline{s_{[1,2n]}} \xrightarrow{\varphi}_{\mathcal{T}} \mathsf{fail}$ and the transition

(12) $\mathsf{fail} \xrightarrow{a_i}_{\mathcal{T}} \mathsf{fail}$ for each $i \in [1, 2n]$.

Let us now define the outgoing transitions of u_I for each non-game-conform $I \subseteq [1, 2n]$:

(1) $u_I \xrightarrow{a_i}_{\mathcal{T}} s_{[1,2n]}$ for each $i \in I$,

(2) $u_I \xrightarrow{a_i}_{\mathcal{T}} u_{I\cup\{i\}}$ for each $i \notin I$ for which $I \cup \{i\}$ is non-game-conform,

(3) $u_I \xrightarrow{a_i}_{\mathcal{T}} s_{I\cup\{i\}}$ for each $i \notin I$ for which $I \cup \{i\}$ is game-conform, and

1. (4) $u_I \xrightarrow{a_i}_{\mathcal{T}} \overline{s_{I\cup\{i\}}}$ for each $i \notin I$ for which $I \cup \{i\}$ is game-conform.

A snapshot of $\mathcal{T}$ is depicted in Figure 9.1.

The RGTRS

Recall that C denotes the set of states of $\mathcal{T}$ and that P denotes the set of proper leaf labels as defined in Section 9.2. We define the ranked alphabet $\Sigma = (\Sigma_i)_{i\in\{0,1,2\}}$ of $\mathcal{R}$ as follows:

- $\Sigma_0 = \{\mathsf{start}\} \cup P \cup C$,
- $\Sigma_1 = \{\mathsf{root}\}$ and

- $\Sigma_2 = \{\bigstar\}$.

We note that the only relevant trees (i.e. configurations) in $\mathcal{T}(\mathcal{R})$ in our reduction whose leafs are labeled with C are *singleton trees*. To $\mathcal{R}$ we add the rewriting rules

- $c \overset{b}{\hookrightarrow} c'$ for each transition $c \overset{b}{\longrightarrow}_{\mathcal{T}} c'$ in $\mathcal{T}$

Furthermore, we add the following leaf rewriting rules, where $\alpha \in \{0, 1\}$:

(1) $\langle I, \alpha\rangle \overset{a_i}{\hookrightarrow} \langle I \cup \{i\}, \alpha\rangle$ for each $i \in [1, 2n] \setminus I$,

(2) $\langle I, \alpha\rangle \overset{a_i}{\hookrightarrow} \langle \bot, \alpha\rangle$ for each $i \in I$, and

(3) $\langle \bot, \alpha\rangle \overset{a_i}{\hookrightarrow} \langle \bot, \alpha\rangle$ for each $i \in I$.

Let us add for each possible $I \subseteq [1, 2n]$ and $k \in [0, 2n-1]$, the following rules to R:

(1) $\mathsf{Combs}_{[1,k]} \overset{a_{k+1}}{\hookrightarrow} \overline{s_{[1,k+1]}}$ if k is odd,

(2) $\mathsf{Combs}_{[1,k]} \overset{a_{k+1}}{\hookrightarrow} s_{[1,k+1]}$ if k is even,

(3) $\mathsf{Combs}_{[1,k]\setminus\{i\}} \overset{a_i}{\hookrightarrow} s_{[1,k]}$ for each $i \in [1, k-2]$,

(4) $\mathsf{Combs}_{[1,k]\setminus\{i\}} \overset{a_i}{\hookrightarrow} \overline{s_{[1,k]}}$ for each $i \in [1, k-2]$,

(5) $\mathsf{Combs}_{I\setminus\{i\}} \overset{a_i}{\hookrightarrow} u_I$ for each $i \in I$ if I is non-game-conform,

(6) $\mathsf{Combs}_I \overset{a_i}{\hookrightarrow} s_{[1,2n]}$ for each $i \in I$,

(7) $\mathsf{Combs}_\varphi \overset{\varphi}{\hookrightarrow} \mathsf{fail}$, and

(8) $\mathsf{Combs}_{\overline{\varphi}} \cup \mathsf{Combs}_\bot \overset{\varphi}{\hookrightarrow} \mathsf{succ}$.

One can easily verify that for each $T \in \mathsf{Combs}_{\overline{\varphi}} \cup \mathsf{Combs}_\bot$ we have $T \sim s_{[1,2n]}$. This following lemma establishes correctness of the construction.

Lemma 9.14 ([76]) *Let $I \subseteq [1, 2n]$. If I is game-conform, then*

(1) $s_I \not\sim \overline{s_I}$,

(2) $\forall T \in \mathsf{Combs}_I$*:* $T \not\sim s_I$ *if, and only if,* $T \models \psi[\nu_T]$*, and*

(3) $\forall T \in \mathsf{Combs}_I$*:* $T \sim \overline{s_I}$ *if, and only if,* $T \models \psi[\nu_T]$. □

Moreover I is non-game-conform, then for each $T \in \mathsf{Combs}_I$ we have $T \sim u_I$.

Finally, in order to realize the initial round (as mentioned above) we add the rules $\mathsf{start} \overset{a_1}{\hookrightarrow} s_{[1,1]}$ and $\mathsf{start} \overset{a_1}{\hookrightarrow} \mathsf{Combs}_{[1,1]}$ to $\mathcal{R}$. Theorem 9.12 follows.

10 List of submitted papers

In total I submit 12 papers published at international conferences, out of which three journal papers have emerged so far.

[10] Michael Benedikt, Stefan Göller, Stefan Kiefer, and Andrzej S. Murawski. Bisimilarity of Pushdown Automata is Nonelementary. In *LICS*. IEEE Computer Society, 2013

[14] Stanislav Böhm and Stefan Göller. Language Equivalence of Deterministic Real-Time One-Counter Automata Is NL-Complete. In *Proc. of MFCS*, volume 6907 of *Lecture Notes in Computer Science*, pages 194–205. Springer, 2011

[15] Stanislav Böhm, Stefan Göller, and Petr Jančar. Bisimilarity of one-counter processes is PSPACE-complete. In *Proc. of CONCUR*, volume 6269 of *Lecture Notes in Computer Science*, pages 177–191. Springer, 2010

[16] Stanislav Böhm, Stefan Göller, and Petr Jančar. Equivalence and regularity of real-time one-counter automata. *Journal of Computer and System Sciences*, 2013. accepted for publication

[17] Stanislav Böhm, Stefan Göller, and Petr Jančar. Equivalence of deterministic one-counter automata is NL-complete. In *Proc. of STOC*. ACM, 2013. to appear

[27] Christopher Broadbent and Stefan Göller. On Bisimilarity of Higher-Order Pushdown Automata: Undecidability at Order Two. In *Proc. of FSTTCS*, volume 18 of *LIPIcs*, pages 160–172. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2012

[73] Stefan Göller, Christoph Haase, Joël Ouaknine, and James Worrell. Model Checking Succinct and Parametric One-Counter Automata. In *Proc. of ICALP (2)*, volume 6199 of *Lecture Notes in Computer Science*, pages 575–586. Springer, 2010

[74] Stefan Göller, Christoph Haase, Joël Ouaknine, and James Worrell. Branching-time model checking of parametric one-counter automata. In *Proc. of FoSSaCS*, volume 7213 of *Lecture Notes in Computer Science*, pages 406–420. Springer, 2012

[75] Stefan Göller and Anthony Widjaja Lin. Refining the Process Rewrite Systems Hierarchy via Ground Tree Rewrite Systems. In *Proc. of CONCUR*, volume 6901 of *Lecture Notes in Computer Science*, pages 543–558. Springer, 2011

[76] Stefan Göller and Anthony Widjaja Lin. The Complexity of Verifying Ground Tree Rewrite Systems. In *Proc. of LICS*, pages 279–288. IEEE Computer Society, 2011

[77] Stefan Göller and Anthony Widjaja Lin. Concurrency Makes Simple Theories Hard. In *Proc. of STACS*, LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2012. to appear

[78] Stefan Göller and Anthony Widjaja Lin. Refining the Process Rewrite Systems Hierarchy via Ground Tree Rewrite Systems. *Transactions on Computational Logic*, 2013. accepted for publication

[79] Stefan Göller and Markus Lohrey. Branching-time model checking of one-counter processes. In *Proc. of STACS*, volume 5 of *LIPIcs*, pages 405–416. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2010

[80] Stefan Göller and Markus Lohrey. Branching-time model checking of one-counter processes and timed automata. *SIAM Journal of Computing*, 2013. to appear

[82] Stefan Göller, Richard Mayr, and Anthony Widjaja To. On the computational complexity of verifying one-counter processes. In *Proc. of LICS*, pages 235–244. IEEE Computer Society Press, 2009

Bibliography

[1] Parosh Aziz Abdulla and Karlis Cerans. Simulation is decidable for one-counter nets (extended abstract). In *Proc. of CONCUR*, volume 1466 of *Lecture Notes in Computer Science*, pages 253–268. Springer, 1998.

[2] Eric Allender, Michal Koucky, Detlef Ronneburger, and Sambuddha Roy. The pervasive reach of resource-bounded kolmogorov complexity in computational complexity theory. *To appear at JCSS*, 2010.

[3] Rajeev Alur and David L. Dill. A theory of timed automata. *Theor. Comput. Sci.*, 126(2):183–235, 1994.

[4] André Arnold and Damian Niwiński. *Rudiments of μ-calculus*, volume 146 of *Studies in Logic and the Foundations of Mathematics*. North-Holland, 2001.

[5] J. C. M. Baeten and W. P. Weijland. *Process Algebra*. Cambridge University Press, 1990.

[6] Jos C. M. Baeten, Jan A. Bergstra, and Jan Willem Klop. Decidability of bisimulation equivalence for processes generating context-free languages. In J. W. de Bakker, A. J. Nijman, and Philip C. Treleaven, editors, *PARLE (2)*, volume 259 of *Lecture Notes in Computer Science*, pages 94–111. Springer, 1987.

[7] José L. Balcázar, Joaquim Gabarró, and Miklos Santha. Deciding Bisimilarity is P-Complete. *Formal Asp. Comput.*, 4(6A):638–648, 1992.

[8] Vince Barany, Erich Grädel, and Sasha Rubin. Automata-based presentations of infinite structures. *London Mathematical Society Lecture Notes Series*, 379, 2011.

[9] Paul W. Beame, Stephen A. Cook, and H. James Hoover. Log depth circuits for division and related problems. *SIAM Journal on Computing*, 15(4):994–1003, 1986.

[10] Michael Benedikt, Stefan Göller, Stefan Kiefer, and Andrzej S. Murawski. Bisimilarity of Pushdown Automata is Nonelementary. In *LICS*. IEEE Computer Society, 2013.

[11] Jan A. Bergstra and Jan Willem Klop. Algebra of communicating processes with abstraction. *Theor. Comput. Sci.*, 37:77–121, 1985.

[12] Piotr Berman and Robert Roos. Learning One-Counter Languages in Polynomial Time (Extended Abstract). In *Proc. of FOCS*, pages 61–67. IEEE, 1987.

[13] Michel Blockelet and Sylvain Schmitz. Model checking coverability graphs of vector addition systems. In *Proc. of MFCS*, volume 6907 of *Lecture Notes in Computer Science*, pages 108–119. Springer, 2011.

[14] Stanislav Böhm and Stefan Göller. Language Equivalence of Deterministic Real-Time One-Counter Automata Is NL-Complete. In *Proc. of MFCS*, volume 6907 of *Lecture Notes in Computer Science*, pages 194–205. Springer, 2011.

[15] Stanislav Böhm, Stefan Göller, and Petr Jančar. Bisimilarity of one-counter processes is PSPACE-complete. In *Proc. of CONCUR*, volume 6269 of *Lecture Notes in Computer Science*, pages 177–191. Springer, 2010.

[16] Stanislav Böhm, Stefan Göller, and Petr Jančar. Equivalence and regularity of real-time one-counter automata. *Journal of Computer and System Sciences*, 2013. accepted for publication.

[17] Stanislav Böhm, Stefan Göller, and Petr Jančar. Equivalence of deterministic one-counter automata is NL-complete. In *Proc. of STOC*. ACM, 2013. to appear.

[18] Ahmed Bouajjani, Javier Esparza, and Oded Maler. Reachability analysis of pushdown automata: Application to model-checking. In Antoni W. Mazurkiewicz and Józef Winkowski, editors, *Proc. of CONCUR*, volume 1243 of *Lecture Notes in Computer Science*, pages 135–150. Springer, 1997.

[19] Ahmed Bouajjani and Peter Habermehl. Constrained properties, semilinear systems, and petri nets. In *Proc. of CONCUR*, volume 1119 of *Lecture Notes in Computer Science*, pages 481–497. Springer, 1996.

[20] Ahmed Bouajjani, Markus Müller-Olm, and Tayssir Touili. Regular symbolic analysis of dynamic networks of pushdown systems. In Martín Abadi and Luca de Alfaro, editors, *CONCUR*, volume 3653 of *Lecture Notes in Computer Science*, pages 473–487. Springer, 2005.

[21] Ahmed Bouajjani and Tayssir Touili. Reachability analysis of process rewrite systems. In Paritosh K. Pandya and Jaikumar Radhakrishnan, editors, *Proc. of FSTTCS*, volume 2914 of *Lecture Notes in Computer Science*, pages 74–87. Springer, 2003.

[22] Patricia Bouyer and François Laroussinie. Model checking timed automata. In Stephan Merz and Nicolas Navet, editors, *Modeling and Verification of Real-Time Systems*, pages 111–140. ISTE Ltd. – John Wiley & Sons, Ltd., January 2008.

[23] Laura Bozzelli. Complexity results on branching-time pushdown model checking. *Theor. Comput. Sci.*, 379(1-2):286–297, 2007.

[24] Laura Bozzelli, Mojmír Kretínský, Vojtech Rehák, and Jan Strejcek. On decidability of LTL model checking for process rewrite systems. *Acta Inf.*, 46(1):1–28, 2009.

[25] Walter S. Brainerd. Tree generating regular systems. *Information and Control*, 14(2):217–231, 1969.

[26] Tomás Brázdil, Václav Brozek, Kousha Etessami, Antonín Kučera, and Dominik Wojtczak. One-counter markov decision processes. In *SODA*, pages 863–874. SIAM, 2010.

[27] Christopher Broadbent and Stefan Göller. On Bisimilarity of Higher-Order Pushdown Automata: Undecidability at Order Two. In *Proc. of FSTTCS*, volume 18 of *LIPIcs*, pages 160–172. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2012.

[28] Christopher H. Broadbent and C.-H. Luke Ong. On global model checking trees generated by higher-order recursion schemes. In *Proc. of FOSSACS*, volume 5504 of *Lecture Notes in Computer Science*, pages 107–121. Springer, 2009.

[29] J. Richard Büchi. Regular canonical systems. *Archiv für Mathematische Logik und Grundlagenforschung*, 6:91–111, 1964.

[30] Olaf Burkart, Didier Caucal, Faron Moller, and Bernhard Steffen. Verification on infinite structures. In Jan A. Bergstra, Alban Ponse, and Scott A. Smolka, editors, *Handbook of process algebra*, pages 545–623. Elsevier, 2001.

[31] Olaf Burkart, Didier Caucal, and Bernhard Steffen. An Elementary Bisimulation Decision Procedure for Arbitrary Context-Free Processes. In *Proc. of MFCS*, volume 969 of *Lecture Notes in Computer Science*, pages 423–433. Springer, 1995.

[32] T. Cachat. Higher order pushdown automata, the Caucal hierarchy of graphs and parity games. In *Proceedings of ICALP*, volume 2719 of *Lecture Notes in Computer Science*, pages 556–569. Springer, 2003.

[33] Thierry Cachat. Uniform solution of parity games on prefix-recognizable graphs. *Electronic Notes Theoretical Computer Science*, 68(6), 2002.

[34] Thierry Cachat and Igor Walukiewicz. The Complexity of Games on Higher Order Pushdown Automata. *CoRR*, abs/0705.0262, 2007.

[35] Jin-Yi Cai and Merrick Furst. PSPACE survives constant-width bottlenecks. *International Journal of Foundations of Computer Science*, 2(1):67–76, 1991.

[36] Arnaud Carayol and Stefan Wöhrle. The Caucal Hierarchy of Infinite Graphs in Terms of Logic and Higher-Order Pushdown Automata. In *FSTTCS*, volume 2914 of *Lecture Notes in Computer Science*, pages 112–123. Springer, 2003.

[37] E. Cardoza, Richard J. Lipton, and Albert R. Meyer. Exponential space complete problems for petri nets and commutative semigroups: Preliminary report. In *STOC*, pages 50–54. ACM, 1976.

[38] D. Caucal. On infinite transition graphs having a decidable monadic theory. *Theoretical Computer Science*, 290(1):79–115, 2002.

[39] Didier Caucal. On the regular structure of prefix rewriting. *Theoretical Computer Science*, 106:61–86, 1992.

[40] Didier Caucal. On infinite transition graphs having a decidable monadic theory. In Friedhelm Meyer auf der Heide and Burkhard Monien, editors, *Proceedings of the 23nd International Colloquium on Automata, Languages and Programming (ICALP'96), Paderborn (Germany)*, number 1099 in Lecture Notes in Computer Science, pages 194–205. Springer, 1996.

[41] Didier Caucal. On infinite transition graphs having a decidable monadic theory. *Theor. Comput. Sci.*, 290(1):79–115, 2003.

[42] Didier Caucal, Dung T. Huynh, and Lu Tian. Deciding Branching Bimiliarity of Normed Context-Free Processes Is in Σ_2^p. *Inf. Comput.*, 118(2):306–315, 1995.

[43] Andrew Chiu, George Davida, and Bruce Litow. Division in logspace-uniform NC^1. *Theoretical Informatics and Applications. Informatique Théorique et Applications*, 35(3):259–275, 2001.

[44] Christian Choffrut and Massimiliano Goldwurm. Timed automata with periodic clock constraints. *Journal of Automata, Languages and Combinatorics*, 5(4):371–404, 2000.

[45] Søren Christensen. *Decidability and Decomposition in Process Algebras*. PhD thesis, Department of Computer Science, The University of Edinburgh, 1993.

[46] Marek Chrobak. Finite automata and unary languages. *Theoretical Computer Science*, 47(3):149–158, 1986.

[47] Edmund M. Clarke and E. Allen Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. In *Logics of Programs*, volume 131 of *Lecture Notes in Computer Science*, pages 52–71. Springer, 1981.

[48] Thomas Colcombet. On families of graphs having a decidable first order theory with reachability. In Peter Widmayer, Francisco Triguero Ruiz, Rafael Morales, Matthew Hennessy, Stephan Eidenbenz, and Ricardo Conejo, editors, *Proceedings of the 29th International Colloquium on Automata, Languages and Programming (ICALP 2002), Malaga (Spain)*, number 2380 in Lecture Notes in Computer Science, pages 98–109. Springer, 2002.

[49] Hubert Comon, Max Dauchet, Remi Gilleron, Christof Löding, Florent Jacquemard, Denis Lugiez, Sophie Tison, and Marc Tommasi. Tree automata techniques and applications. Available on: `http://www.grappa.univ-lille3.fr/tata`, 2007. release October, 12th 2007.

[50] Jean-Luc Coquidé, Max Dauchet, Rémi Gilleron, and Sándor Vágvölgyi. Bottom-up tree pushdown automata: Classification and connection with rewrite systems. *Theor. Comput. Sci.*, 127(1):69–98, 1994.

[51] Costas Courcoubetis and Mihalis Yannakakis. Minimum and maximum delay problems in real-time systems. *Formal Methods in System Design*, 1(4):385–415, 1992.

[52] Wojciech Czerwinski, Piotr Hofman, and Slawomir Lasota. Decidability of branching bisimulation on normed commutative context-free processes. In *Proc. of CONCUR*, volume 6901 of *Lecture Notes in Computer Science*, pages 528–542. Springer, 2011.

[53] Wojciech Czerwinski and Slawomir Lasota. Fast equivalence-checking for normed context-free processes. In *Proc. of FSTTCS*, volume 8 of *LIPIcs*, pages 260–271. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2010.

[54] W. Damm and A. Goerdt. An automata-theoretical characterization of the OI-hierarchy. *Information and Control*, 71(1-2):1–32, October 1986.

[55] Philippe Darondeau, Stéphane Demri, Roland Meyer, and Christophe Morvan. Petri net reachability graphs: Decidability status of fo properties. In *Proc. of FSTTCS*, volume 13 of *LIPIcs*, pages 140–151. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2011.

[56] Max Dauchet and Sophie Tison. The theory of ground rewrite systems is decidable. In *Proceedings of the 5th Annual IEEE Symposium on Logic in Computer Science (LICS '90)*, pages 242–256. IEEE Computer Society Press, 1990.

[57] Anuj Dawar, Martin Grohe, Stephan Kreutzer, and Nicole Schweikardt. Model Theory Makes Formulas Large. In *Proc. of ICALP*, volume 4596 of *Lecture Notes in Computer Science*. Springer, 2007.

[58] Stéphane Demri. On selective unboundedness of vass. In *Proc. of INFINITY*, volume 39 of *EPTCS*, pages 1–15, 2010.

[59] E. Allen Emerson and Charanjit S. Jutla. Tree automata, mu-calculus and determinacy (extended abstract). In *FOCS*, pages 368–377. IEEE Computer Society, 1991.

[60] Javier Esparza. On the Decidabilty of Model Checking for Several mu-calculi and Petri Nets. In *Proceedings of the 19th International Colloquium on Trees in Algebra and Programming*, volume 787 of *Lecture Notes in Computer Science*, pages 115–129. Springer, 1994.

[61] Javier Esparza. Petri Nets, Commutative Context–Free Grammars, and Basic Parallel Processes. *Fundamenta Informatica*, 30:23–41, 1997.

[62] Javier Esparza and Astrid Kiehn. On the Model Checking Problem for Branching Time Logics and Basic Parallel Processes. In *CAV*, volume 939 of *Lecture Notes in Computer Science*, pages 353–366. Springer, 1995.

[63] Javier Esparza and Andreas Podelski. Efficient Algorithms for pre^* and $post^*$ on Interprocedural Parallel Flow Graphs. In *Proc. of POPL*, pages 1–11. ACM, 2000.

[64] Kousha Etessami and Mihalis Yannakakis. Recursive markov decision processes and recursive stochastic games. In *Proceesing of the 32nd International Colloquium on Automata, Languages and Programming (ICALP 2005)*, number 3580 in Lecture Notes in Computer Science, pages 891–903, 2005.

[65] John Fearnley and Marcin Jurdzinski. Reachability in two-clock timed automata is pspace-complete. *CoRR*, abs/1302.3109, 2013.

[66] Ingo Felscher and Wolfgang Thomas. Compositionality and Reachability with Conditions on Path Lengths. *Int. J. Found. Comput. Sci.*, 20(5), 2009.

[67] Oliver Friedmann and Martin Lange. Solving parity games in practice. In *ATVA*, volume 5799 of *Lecture Notes in Computer Science*, pages 182–196. Springer, 2009.

[68] Yuxi Fu. Checking Equality and Regularity for Normed BPA with Silent Moves. In *Proc. of ICALP*, Lecture Notes in Computer Science. Springer, 2013. to appear.

[69] R.J. van Glabbeek. The linear time – branching time spectrum I; the semantics of concrete, sequential processes. In J.A. Bergstra, A. Ponse, and S.A. Smolka, editors, *Handbook of Process Algebra*, chapter 1, pages 3–99. Elsevier, 2001.

[70] Rob J. van Glabbeek. The linear time – branching time spectrum II; the semantics of sequential systems with silent moves (extended abstract). In E. Best, editor, *Proceedings of the 4th International Conference on Concurrency Theory (CONCUR'93), Hildesheim (Germany)*, number 715 in Lecture Notes in Computer Science, pages 66–81. Springer, 1993.

[71] Stefan Göller. Reachability on prefix-recognizable graphs. *Inf. Process. Lett.*, 108(2):71–74, 2008.

[72] Stefan Göller. *The Computational Complexity of Propositional Dynamic Logics*. PhD Thesis, University of Leipzig, 2008.

[73] Stefan Göller, Christoph Haase, Joël Ouaknine, and James Worrell. Model Checking Succinct and Parametric One-Counter Automata. In *Proc. of ICALP (2)*, volume 6199 of *Lecture Notes in Computer Science*, pages 575–586. Springer, 2010.

[74] Stefan Göller, Christoph Haase, Joël Ouaknine, and James Worrell. Branching-time model checking of parametric one-counter automata. In *Proc. of FoSSaCS*, volume 7213 of *Lecture Notes in Computer Science*, pages 406–420. Springer, 2012.

[75] Stefan Göller and Anthony Widjaja Lin. Refining the Process Rewrite Systems Hierarchy via Ground Tree Rewrite Systems. In *Proc. of CONCUR*, volume 6901 of *Lecture Notes in Computer Science*, pages 543–558. Springer, 2011.

[76] Stefan Göller and Anthony Widjaja Lin. The Complexity of Verifying Ground Tree Rewrite Systems. In *Proc. of LICS*, pages 279–288. IEEE Computer Society, 2011.

[77] Stefan Göller and Anthony Widjaja Lin. Concurrency Makes Simple Theories Hard. In *Proc. of STACS*, LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2012. to appear.

[78] Stefan Göller and Anthony Widjaja Lin. Refining the Process Rewrite Systems Hierarchy via Ground Tree Rewrite Systems. *Transactions on Computational Logic*, 2013. accepted for publication.

[79] Stefan Göller and Markus Lohrey. Branching-time model checking of one-counter processes. In *Proc. of STACS*, volume 5 of *LIPIcs*, pages 405–416. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2010.

[80] Stefan Göller and Markus Lohrey. Branching-time model checking of one-counter processes and timed automata. *SIAM Journal of Computing*, 2013. to appear.

[81] Stefan Göller, Markus Lohrey, and Carsten Lutz. PDL with Intersection and Converse Is 2 EXP-Complete. In *Proc. of FoSSaCS*, volume 4423 of *Lecture Notes in Computer Science*, pages 198–212. Springer, 2007.

[82] Stefan Göller, Richard Mayr, and Anthony Widjaja To. On the computational complexity of verifying one-counter processes. In *Proc. of LICS*, pages 235–244. IEEE Computer Society Press, 2009.

[83] Christoph Haase, Stephan Kreutzer, Joël Ouaknine, and James Worrell. Reachability in Succinct and Parametric One-Counter Automata. In *Proc. of CONCUR*, volume 5710 of *LNCS*, pages 369–383. Springer, 2009.

[84] Christoph Haase, Joël Ouaknine, and James Worrell. On the relationship between reachability problems in timed and counter automata. In *Proc. of RP*, volume 7550 of *Lecture Notes in Computer Science*, pages 54–65. Springer, 2012.

[85] Peter Habermehl. On the complexity of the linear-time mu -calculus for petri-nets. In *Proc. of ICATPN*, volume 1248 of *Lecture Notes in Computer Science*, pages 102–116. Springer, 1997.

[86] M. H. T. Hack. *Decidability Questions for Petri Nets*. PhD thesis, MIT, 1976.

[87] Matthew Hague, Andrzej S. Murawski, C.-H. Luke Ong, and Olivier Serre. Collapsible pushdown automata and recursion schemes. In *Proc. of LICS*, pages 452–461. IEEE Computer Society, 2008.

[88] Matthew Hague and C.-H. Luke Ong. Symbolic backwards-reachability analysis for higher-order pushdown systems. *Logical Methods in Computer Science*, 4(4), 2008.

[89] Matthew Hague and Anthony Widjaja To. The Complexity of Model Checking (Collapsible) Higher-Order Pushdown Systems. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2010)*, volume 8 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 228–239. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2010.

[90] Ulrich Hertrampf, Clemens Lautemann, Thomas Schwentick, Heribert Vollmer, and Klaus W. Wagner. On the power of polynomial time bit-reductions. In *Proceedings*

of the Eighth Annual Structure in Complexity Theory Conference, pages 200–207. IEEE Computer Society Press, 1993.

[91] William Hesse, Eric Allender, and David A. Mix Barrington. Uniform constant-depth threshold circuits for division andd iterated multiplication. *Journal of Computer and System Sciences*, 65:695–716, 2002. `http://www.cs.umass.edu/~whesse/`.

[92] Yoram Hirshfeld. Petri nets and the equivalence problem. In *CSL*, volume 832 of *Lecture Notes in Computer Science*, pages 165–174. Springer, 1993.

[93] Yoram Hirshfeld and Mark Jerrum. Bisimulation equivanlence is decidable for normed process algebra. In *Proc. of ICALP*, volume 1644 of *Lecture Notes in Computer Science*, pages 412–421. Springer, 1999.

[94] Yoram Hirshfeld, Mark Jerrum, and Faron Moller. A polynomial algorithm for deciding bisimilarity of normed context-free processes. *Theoretical Computer Science*, 158(1&2):143–159, 1996.

[95] Piotr Hofman, Richard Mayr, and Patrick Totzke. Decidability of Weak Simulation on One-counter Nets. In *Proc. of LICS*, page ??? IEEE, 2013. to appear.

[96] Piotr Hofman and Patrick Totzke. Approximating Weak Bisimilarity of Basic Parallel Processes. In *EXPRESS/SOS*, volume 89 of *EPTCS*, pages 99–113, 2012.

[97] David Janin and Igor Walukiewicz. On the Expressive Completeness of the Propositional mu-Calculus with Respect to Monadic Second Order Logic. In *Proc. of CONCUR*, volume 1119 of *Lecture Notes in Computer Science*, pages 263–277. Springer, 1996.

[98] P. Jančar and Z. Sawa. A note on emptiness for alternating finite automata with a one-letter alphabet. *Information Processing Letters*, 104(5):164–167, 2007.

[99] Petr Jančar. Undecidability of Bisimilarity for Petri Nets and Some Related Problems. *Theor. Comput. Sci.*, 148(2):281–301, 1995.

[100] Petr Jančar. Decidability of Bisimilarity for One-Counter Processes. *Information Computation*, 158(1):1–17, 2000.

[101] Petr Jančar. Strong Bisimilarity on Basic Parallel Processes is PSPACE-complete. In *Proc. of LICS*, pages 218–227. IEEE Computer Society, 2003.

[102] Petr Jančar. Bisimilarity on Basic Process Algebra is in 2-ExpTime (an explicit proof). *CoRR*, abs/1207.2479, 2012.

[103] Petr Jančar. Decidability of DPDA Language Equivalence via First-Order Grammars. In *LICS*, pages 415–424. IEEE, 2012.

[104] Petr Jančar. Finiteness up to bisimilarity is decidable for pushdown processes. *CoRR*, abs/1305.0516, 2013.

[105] Petr Jančar, Javier Esparza, and Faron Moller. Petri nets and regular processes. *J. Comput. Syst. Sci.*, 59(3):476–503, 1999.

[106] Petr Jančar, Antonín Kucera, and Faron Moller. Simulation and bisimulation over one-counter processes. In *Proc. of STACS*, volume 1770 of *Lecture Notes in Computer Science*, pages 334–345, 2000.

[107] Petr Jančar, Antonín Kučera, and Richard Mayr. Deciding bisimulation-like equivalences with finite-state processes. *Theor. Comput. Sci.*, 258(1-2):409–433, 2001.

[108] Petr Jančar, Antonín Kučera, Faron Moller, and Zdenek Sawa. DP lower bounds for equivalence-checking and model-checking of one-counter automata. *Inf. Comput.*, 188(1):1–19, 2004.

[109] Petr Jančar, Faron Moller, and Zdenek Sawa. Simulation Problems for One-Counter Machines. In *Proc. of SOFSEM*, volume 1725 of *Lecture Notes in Computer Science*, pages 404–413. Springer, 1999.

[110] Petr Jančar and Jirí Srba. Undecidability of bisimilarity by defender's forcing. *J. ACM*, 55(1), 2008.

[111] Marcin Jurdzinski. Deciding the winner in parity games is in up cap co-up. *Inf. Process. Lett.*, 68(3):119–124, 1998.

[112] Marcin Jurdzinski, Mike Paterson, and Uri Zwick. A Deterministic Subexponential Algorithm for Solving Parity Games. *SIAM J. Comput.*, 38(4):1519–1532, 2008.

[113] Paris C. Kanellakis and Scott A. Smolka. CCS expressions, finite state processes, and three problems of equivalence. *Information and Computation*, 86(1):43–68, May 1990.

[114] Stefan Kiefer. BPA bisimilarity is EXPTIME-hard. *Inf. Process. Lett.*, 113(4):101–106, 2013.

[115] Teodor Knapik, Damian Niwinski, and Pawel Urzyczyn. Higher-Order Pushdown Trees Are Easy. In *FoSSaCS*, volume 2303 of *Lecture Notes in Computer Science*, pages 205–222. Springer, 2002.

[116] Naoki Kobayashi. Types and higher-order recursion schemes for verification of higher-order programs. In *Proc. of POPL*, pages 416–428. ACM, 2009.

[117] S. R. Kosaraju. Decidability of reachability in vector addition systems. In *14th Annual Symposium on Theory of Computing*, pages 267–281. ACM Press, 1982.

[118] O. Kupferman, N. Piterman, and M. Y. Vardi. Model Checking Linear Properties of Prefix-Recognizable Systems. In *Proceedings of the 14th International Conference on Computer Aided Verification*, volume 2404 of *Lecture Notes in Computer Science*, pages 371–385. Springer, 2002.

[119] Orna Kupferman and Moshe Y. Vardi. Weak alternating automata and tree automata emptiness. In *STOC '98: Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 224–233, New York, NY, USA, 1998. ACM Press.

[120] Orna Kupferman and Moshe Y. Vardi. An Automata-Theoretic Approach to Reasoning about Infinite-State Systems. In *Proceedings of the 12th International Conference on Computer Aided Verification*, number 1855 in Lecture Notes in Computer Science, pages 36–52. Springer, 2000.

[121] Antonín Kučera and Richard Mayr. On the complexity of checking semantic equivalences between pushdown processes and finite-state processes. *Information and Computation*, 208(7):772–796, 2010.

[122] Antonín Kučera and Ph. Schnoebelen. A general approach to comparing infinite-state systems with their finite-state specifications. *Theor. Comput. Sci.*, 358(2-3):315–333, 2006.

[123] M. Lange. Model Checking Propositional Dynamic Logic with All Extras. *Journal of Applied Logic*, 4(1):39–49, 2005.

[124] François Laroussinie, Nicolas Markey, and Ph. Schnoebelen. Efficient timed model checking for discrete-time systems. *Theor. Comput. Sci.*, 353(1-3):249–271, 2006.

[125] Jérôme Leroux. The general vector addition system reachability problem by presburger inductive invariants. In *LICS*, pages 4–13. IEEE Computer Society, 2009.

[126] Leonid Libkin. *Elements of Finite Model Theory*. Springer, 2004.

[127] Orna Lichtenstein and Amir Pnueli. Checking that finite state concurrent programs satisfy their linear specification. In *POPL*, pages 97–107. ACM Press, 1985.

[128] Christof Löding. *Infinite Graphs Generated by Tree Rewriting*. PhD thesis, RWTH Aachen, 2003.

[129] Shan Lu, Soyeon Park, Eunsoo Seo, and Yuanyuan Zhou. Learning from mistakes: a comprehensive study on real world concurrency bug characteristics. In *ASPLOS*, pages 329–339. ACM, 2008.

[130] Denis Lugiez and Ph. Schnoebelen. Decidable first-order transition logics for PA-processes. *Inf. Comput.*, 203(1):75–113, 2005.

[131] Denis Lugiez and Philippe Schnoebelen. The regular viewpoint on PA-processes. *Theor. Comput. Sci.*, 274(1-2):89–115, 2002.

[132] Janos A. Makowsky. Algorithmic aspects of the Feferman-Vaught theorem. *Annals of Pure and Applied Logic*, 126(1–3):159–213, 2004.

[133] Andrew Martinez. Efficient computation of regular expressions from unary NFAs. In *Workshop on Descriptional Complexity of Formal Systems*, pages 216–230, 2002.

[134] A N Maslov. Multilevel Stack Automata. *Problems of Information Transmission*, (12):38–43, 1976.

[135] Yuri Matiyasevich. Enumerable sets are Diophantine. *Soviet Math. Dokl.*, 11:354–357, 1970.

[136] Ernst W. Mayr. An algorithm for the general Petri net reachability problem. *SIAM Journal on Computing*, 13:441–460, 1984.

[137] Richard Mayr. Weak Bisimulation and Model Checking for Basic Parallel Processes. In *FSTTCS*, volume 1180 of *Lecture Notes in Computer Science*, pages 88–99. Springer, 1996.

[138] Richard Mayr. Model Checking PA-Processes. In *Proc. CONCUR*, volume 1243 of *Lecture Notes in Computer Science*, pages 332–346. Springer, 1997.

[139] Richard Mayr. *Decidability and Complexity of Model Checking Problems for Infinite-State Systems*. PhD thesis, TU-Munich, 1998.

[140] Richard Mayr. Process Rewrite Systems. *Information and Computation*, 156(1):264–286, 2000.

[141] Richard Mayr. Decidability of model checking with the temporal logic EF. *Theoretical Computer Science*, 256(1-2):31–62, 2001.

[142] Richard Mayr. Undecidability of weak bisimulation equivalence for 1-counter processes. In *Proc. of ICALP*, volume 2719 of *Lecture Notes in Computer Science*, pages 570–583, 2003.

[143] Robin Milner. *Communication and Concurrency*. International Series in Computer Science. Prentice Hall, 1989.

[144] Robin Milner. *Communication and Concurrency*. Prentice Hall, 1989.

[145] Faron Moller and Alexander Moshe Rabinovich. Counting on CTL*: on the expressive power of monadic path logic. *Inf. Comput.*, 184(1):147–159, 2003.

[146] D. E. Muller and P. E. Schupp. The theory of ends, pushdown automata, and second-order logic. *Theor. Comput. Sci.*, 37:51–75, 1985.

[147] G. Naves. Accessibilité dans les automates temporisé à deux horloges. Memoire de master 2, ENS Cachan (France), 2006.

[148] C.-H. Luke Ong and Steven James Ramsay. Verifying higher-order functional programs with pattern-matching algebraic data types. In *Proc. of POPL*, pages 587–598. ACM, 2011.

[149] Michio Oyamaguchi. The equivalence problem for real-time DPDAs. *J. ACM*, 34:731–760, 1987.

[150] Robert Paige and Robert E. Tarjan. Three partition refinement algorithms. *SIAM Journal on Computing*, 16(6):973–989, December 1987.

[151] Christos H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.

[152] Pawel Parys. Collapse operation increases expressive power of deterministic higher order pushdown automata. In *Proc. of STACS*, volume 9 of *LIPIcs*, pages 603–614. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2011.

[153] Amir Pnueli. The temporal logic of programs. In *Proc. of FOCS*, pages 46–57. IEEE Computer Society, 1977.

[154] Emil Post. A variant of a recursively unsolvable proble. *Bulletin of the American Mathematical Society*, 52(4):264–268, 1946.

[155] Shaz Qadeer and Jakob Rehof. Context-bounded model checking of concurrent software. In *TACAS*, volume 3440 of *Lecture Notes in Computer Science*, pages 93–107. Springer, 2005.

[156] Alexander Rabinovich. On compositionality and its limitations. *ACM Trans. Comput. Log.*, 8(1), 2007.

[157] Charles Rackoff. Relativized questions involving probabilistic algorithms. In *Proc. of STOC*, pages 338–342. ACM, 1978.

[158] Julia Robinson. Definability and Decision Problems in Arithmetic. *J. Symbolic Logic*, 14(2):98–114, 1949.

[159] Robert Roos. *Deciding Equivalence of Deterministic One-Counter Automata in Polynomial Time with Applications to Learning*. PhD thesis, The Pennsylvania State University, 1988.

[160] Keijo Ruohonen. Reversible machines and post's correspondence problem for biprefix morphisms. *Elektronische Informationsverarbeitung und Kybernetik*, 21(12):579–595, 1985.

[161] Sven Schewe. Solving parity games in big steps. In *FSTTCS*, volume 4855 of *Lecture Notes in Computer Science*, pages 449–460. Springer, 2007.

[162] Ph. Schnoebelen. Oracle circuits for branching-time model checking. In *ICALP*, volume 2719 of *Lecture Notes in Computer Science*, pages 790–801. Springer, 2003.

[163] Alexander Schrijver. *Theory of linear and integer programming*. Wiley-Interscience series in discrete mathematics and optimization. Wiley, 1999.

[164] Géraud Sénizergues. L(A)=L(B)? decidability results from complete formal systems. *Theor. Comput. Sci.*, 251(1-2):1–166, 2001.

[165] Géraud Sénizergues. L(A)=L(B)? A simplified decidability proof. *Theor. Comput. Sci.*, 281(1-2):555–608, 2002.

[166] Géraud Sénizergues. The Equivalence Problem for t-Turn DPDA Is Co-NP. In *Proc. of ICALP*, volume 2719 of *Lecture Notes in Computer Science*, pages 478–489. Springer, 2003.

[167] Géraud Sénizergues. The bisimulation problem for equational graphs of finite out-degree. *SIAM J. Comput.*, 34(5):1025–1106, 2005.

[168] O. Serre. Parity Games Played on Transition Graphs of One-Counter Processes. In *Proceedings of FoSSaCS*, volume 3921 of *Lecture Notes in Computer Science*, pages 337–351. Springer, 2006.

[169] Olivier Serre. Parity games played on transition graphs of one-counter processes. In L. Aceto and A. Ingólfsdóttir, editors, *Proc. of FOSSACS*, number 3921 in Lecture Notes in Computer Science. Springer, 2006.

[170] M. Sipser. *Introduction to the theory of computation*. PWS Publishing Company, 1997.

[171] Holger Spakowski and Jörg Vogel. Theta2p-completeness: A classical approach for new results. In *Proc. of FST&TCS 2000*, volume 1974 of *LNCS*, pages 348–360. Springer, 2000.

[172] Jirí Srba. Strong bisimilarity and regularity of basic parallel processes is pspace-hard. In *Proc. of STACS*, volume 2285 of *Lecture Notes in Computer Science*, pages 535–546. Springer, 2002.

[173] Jirí Srba. Undecidability of Weak Bisimilarity for PA-Processes. In *Developments in Language Theory*, volume 2450 of *Lecture Notes in Computer Science*, pages 197–208. Springer, 2002.

[174] Jirí Srba. *Roadmap of Infinite results*, volume Vol 2: Formal Models and Semantics. World Scientific Publishing Co., 2004. Updated version is available at `http://www.brics.dk/~srba/roadmap/`.

[175] Jirí Srba. Beyond Language Equivalence on Visibly Pushdown Automata. *Logical Methods in Computer Science*, 5(1:2), 2009.

[176] Colin Stirling. Lokal model checking games. In *Proc. of CONCUR*, volume 962 of *Lecture Notes in Computer Science*, pages 1–11. Springer, 1995.

[177] Colin Stirling. The joys of bisimulation. In *Mathematical Foundations of Computer Science 1998, 23rd International Symposium, MFCS'98, Brno, Czech Republic, August 24-28, 1998, Proceedings*, volume 1450 of *Lecture Notes in Computer Science*, pages 142–151. Springer, 1998.

[178] Colin Stirling. Deciding DPDA Equivalence Is Primitive Recursive. In *Proc. of ICALP*, volume 2380 of *Lecture Notes in Computer Science*, pages 821–832. Springer, 2002.

[179] Colin Stirling. Second-order simple grammars. In *CONCUR*, volume 4137 of *Lecture Notes in Computer Science*, pages 509–523. Springer, 2006.

[180] Larry J. Stockmeyer. *The complexity of decision problems in automata and logic*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, 1974.

[181] Wolfgang Thomas. On the ehrenfeucht-fraïssé game in theoretical computer science. In *Proc. of TAPSOFT*, volume 668 of *Lecture Notes in Computer Science*, pages 559–568. Springer, 1993.

[182] Anthony Widjaja To. Model Checking FO(R) over One-Counter Processes and beyond. In *Proc. of CSL*, volume 5771 of *Lecture Notes in Computer Science*, pages 485–499. Springer, 2009.

[183] Anthony Widjaja To. Unary finite automata vs. arithmetic progressions. *Inf. Process. Lett.*, 109(17):1010–1014, 2009.

[184] Anthony Widjaja To. *Model Checking Infinite-State Systems: Generic and Specific Approaches*. PhD thesis, LFCS, School of Informatics, University of Edinburgh, 2010.

[185] Anthony Widjaja To and Leonid Libkin. Algorithmic Metatheorems for Decidable LTL Model Checking over Infinite Systems. In C.-H. Luke Ong, editor, *Proc. of FOSSACS*, volume 6014 of *Lecture Notes in Computer Science*, pages 221–236. Springer, 2010.

[186] Leslie G. Valiant and Mike Paterson. Deterministic one-counter automata. *J. Comput. Syst. Sci.*, 10(3):340–350, 1975.

[187] Johan van Benthem. *Modal Correspondence Theory*. PhD thesis, University of Amsterdam, 1976.

[188] Rob J. van Glabbeek. The linear time-branching time spectrum (extended abstract). In *CONCUR*, volume 458 of *Lecture Notes in Computer Science*, pages 278–297. Springer, 1990.

[189] Rob J. van Glabbeek and W. P. Weijland. Branching time and abstraction in bisimulation semantics. *J. ACM*, 43(3):555–600, 1996.

[190] M. Y. Vardi. Reasoning about The Past with Two-Way Automata. In *Proceedings of the 25th International Colloquium on Automata, Languages and Programming*, volume 1443, pages 628–641, London, UK, 1998. Springer.

[191] Moshe Y. Vardi. The Complexity of Relational Query Languages. In *Proceedings of the 14th Annual ACM Symposium on Theory of Computing*, pages 137–146. ACM Press, 1982.

[192] Moshe Y. Vardi. An automata-theoretic approach to linear temporal logic. In *Banff Higher Order Workshop*, volume 1043 of *Lecture Notes in Computer Science*, pages 238–266. Springer, 1995.

[193] Heribert Vollmer. A generalized quantifier concept in computational complexity theory. Technical report, arXiv.org, 1998. http://arxiv.org/abs/cs.CC/9809115.

[194] Klaus W. Wagner. More complicated questions about maxima and minima, and some closures of NP. *Theor. Comput. Sci.*, 51(1-2):53–80, 1987.

[195] Igor Walukiewicz. Difficult configurations – on the complexity of LTrL. In Kim Guldstrand Larsen, Sven Skyum, and Glynn Winskel, editors, *Proceedings of the 25th International Colloquium on Automata, Languages and Programming (ICALP 98), Aalborg (Denmark)*, number 1443 in Lecture Notes in Computer Science, pages 140–151. Springer, 1998.

[196] Igor Walukiewicz. Model Checking CTL Properties of Pushdown Systems. In *Proceedings of the 20th Conference on Foundations of Software Technology and Theoretical Computer Science*, volume 1974 of *Lecture Notes in Computer Science*, pages 127–138. Springer, 2000.

[197] Igor Walukiewicz. Pushdown Processes: Games and Model-Checking. *Inf. Comput.*, 164(2):234–263, 2001.

[198] Igor Walukiewicz. Monadic second-order logic on tree-like structures. *Theoretical Computer Science*, 275(1–2):311–346, 2002.

[199] Stefan Wöhrle and Wolfgang Thomas. Model checking synchronized products of infinite transition systems. *Logical Methods in Computer Science*, 3(4), 2007.

[200] Hsu-Chun Yen. Complexity Analysis of Some Verification Problems for One-Counter Machines. unpublished manuscript.