

Model Checking Infinite-State Systems: Generic and Specific Approaches

Anthony Widjaja To



Doctor of Philosophy
Laboratory for Foundations of Computer Science
School of Informatics
University of Edinburgh
2010

Abstract

Model checking is a fully-automatic formal verification method that has been extremely successful in validating and verifying safety-critical systems in the past three decades. In the past fifteen years, there has been a lot of work in extending many model checking algorithms over finite-state systems to *finitely representable infinite-state systems*. Unlike in the case of finite systems, decidability can easily become a problem in the case of infinite-state model checking.

In this thesis, we present *generic* and *specific* techniques that can be used to derive decidability with near-optimal computational complexity for various model checking problems over infinite-state systems. Generic techniques and specific techniques primarily differ in the way in which a decidability result is derived. Generic techniques is a “top-down” approach wherein we start with a Turing-powerful formalism for infinite-state systems (in the sense of being able to generate the computation graphs of Turing machines up to isomorphisms), and then impose semantic restrictions whereby the desired model checking problem becomes decidable. In other words, to show that a *subclass* of the infinite-state systems that is generated by this formalism is decidable with respect to the model checking problem under consideration, we will simply have to prove that this subclass satisfies the semantic restriction. On the other hand, specific techniques is a “bottom-up” approach in the sense that we restrict to a non-Turing powerful formalism of infinite-state systems *at the outset*. The main benefit of generic techniques is that they can be used as *algorithmic metatheorems*, i.e., they can give unified proofs of decidability of various model checking problems over infinite-state systems. Specific techniques are more flexible in the sense they can be used to derive decidability or optimal complexity when generic techniques fail.

In the first part of the thesis, we adopt word/tree automatic transition systems as a generic formalism of infinite-state systems. Such formalisms can be used to generate many interesting classes of infinite-state systems that have been considered in the literature, e.g., the computation graphs of counter systems, Turing machines, push-down systems, prefix-recognizable systems, regular ground-tree rewrite systems, PA-processes, order-2 collapsible pushdown systems. Although the generality of these formalisms make most interesting model checking problems (even safety) undecidable, they are known to have nice closure and algorithmic properties. We use these nice properties to obtain several algorithmic metatheorems over word/tree automatic systems, e.g., for deriving decidability of various model checking problems including recurrent reachability, and Linear Temporal Logic (LTL) with complex fairness con-

straints. These algorithmic metatheorems can be used to *uniformly* prove decidability with optimal (or near-optimal) complexity of various model checking problems over many classes of infinite-state systems that have been considered in the literature. In fact, many of these decidability/complexity results were not previously known in the literature.

In the second part of the thesis, we study various model checking problems over subclasses of counter systems that were already known to be decidable. In particular, we consider reversal-bounded counter systems (and their extensions with discrete clocks), one-counter processes, and networks of one-counter processes. We shall derive optimal complexity of various model checking problems including: model checking LTL, EF-logic, and first-order logic with reachability relations (and restrictions thereof). In most cases, we obtain a single/double exponential reduction in the previously known upper bounds on the complexity of the problems.

Acknowledgements

First and foremost, I would like to express my deepest gratitude to my supervisor Leonid Libkin for his teaching, guidance and sound advice throughout my PhD studies. I have learnt a lot from him about many aspects of logic and automata, which I would unlikely have looked at otherwise. His teaching and discussions have always been a source of inspirations. Leonid has also helped me with many academic and personal issues that I encountered throughout my PhD studies. I cannot thank him enough for all these helps.

I would also like to thank my co-supervisor Richard Mayr for his guidance and his inspiring introduction to the area of verification of infinite-state systems. I have lost count of how many hours of useful discussions about research which we have had in the last two years.

I am honored to have Luke Ong and Colin Stirling as my PhD examiners. Thank you for going through the thesis and providing a lot of useful feedback.

I would also like to thank Stefan Göller and Matthew Hague for the fruitful collaborations that we have had throughout my PhD studies. I look forward to working closely together again in the near future. In addition, I am grateful to many other colleagues presently or previously at the School of Informatics with whom I have had many useful discussions: Shunichi Amano, Lorenzo Clemente, Claire David, Christophe Dubach, Kousha Etessami, Floris Geerts, Luis Fabricio Wanderley Goes, Julian Gutierrez, Xibei Jia, Shuai Ma, Filip Murlak, Vasileios Porpodas, Juan Reutter, Rahul Santhanam, Karthik T. Sundararajan, Tony Tan, and Yinghui Wu. I would also like to thank colleagues from other institutes with whom I have had useful research discussions throughout my PhD studies: Vince Barany, Pablo Barcelo, Eryk Kopczynski, Jerome Leroux, Christof Löding, Christophe Morvan, and Sanming Zhou.

I would also like to acknowledge many teachers and colleagues from Australia who have been a great source of inspiration. In particular, I thank James Bailey, Harald Sondergaard, and Sanming Zhou for introducing me to theoretical computer science. I am also indebted to my mentors Mike Ciavarella and Benjamin Rubinstein for their personal and academic advice.

My PhD studies have been made possible by Overseas Research Students Award Scheme and EPSRC grant E005039. Thank you for the generous support.

Last but not least, I would like to thank my family and my girlfriend who have supported me personally throughout my PhD studies. This thesis is dedicated for you.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(Anthony Widjaja To)

Table of Contents

1	Introduction	1
1.1	Specific approaches	4
1.2	Generic approaches	8
1.3	Contributions	9
1.4	Organizations	12
2	Preliminaries	13
2.1	General notations	13
2.2	Automata theory	14
2.2.1	Automata over finite words	15
2.2.2	Automata over ω -words	22
2.2.3	Automata over trees	23
2.2.4	Automata over infinite trees	29
2.2.5	Pushdown automata and context-free grammars	30
2.2.6	Parikh's Theorem and semilinear sets	31
2.3	Computability and complexity theory	33
2.3.1	Computability	33
2.3.2	Complexity theory	34
2.4	Structures and transition systems	37
2.5	Logics and properties	38
2.5.1	Safety, liveness, and fairness	39
2.5.2	Reachability and recurrent reachability	40
2.5.3	FO: First-order logic	41
2.5.4	FO _{REG} (Reach): FO with regular reachability	44
2.5.5	HM-logic: Hennessy-Milner Logic	45
2.5.6	EF _{REG} -logic: HM-logic with regular reachability	46
2.5.7	CTL: Computation Tree logic	47

2.5.8	LTL: Linear Temporal Logic	48
2.5.9	Other logics	49
2.5.10	Model checking complexity	50
I	Generic Approaches: Algorithmic Metatheorems	53
3	Word/Tree-automatic Systems	55
3.1	Word-automatic systems	56
3.1.1	Basic definitions	56
3.1.2	Examples	58
3.1.3	Basic closure and algorithmic results	62
3.1.4	Negative results	67
3.2	Tree-automatic systems	69
3.2.1	Basic definitions	69
3.2.2	Examples	71
3.2.3	Basic closure and algorithmic results	73
3.2.4	Negative results	75
3.3	Other generic frameworks	75
3.3.1	Length-preserving word-automatic transition systems	75
3.3.2	Presburger transition systems	77
3.3.3	Rational transition systems	78
3.3.4	ω -word automatic transition systems	80
4	Algorithmic metatheorems for recurrent reachability	81
4.1	The word-automatic case	83
4.1.1	Main theorem	83
4.1.2	Proof of the main theorem	84
4.1.3	A small witness for recurrent reachability	91
4.1.4	Two appetizer examples	92
4.2	The tree-automatic case	95
4.2.1	Main theorem	95
4.2.2	Preliminaries	96
4.2.3	Proof of the main theorem	97
4.2.4	An appetizer example	105
4.3	Generalized Büchi conditions	106

4.3.1	Word-automatic systems	107
4.3.2	Tree-automatic systems	107
4.3.3	Applications	108
4.4	Recurrent reachability via approximation	110
5	Algorithmic metatheorems for logic model checking	113
5.1	Model checking LTL	115
5.2	Model checking LTL fragments	120
5.2.1	Almost linear Büchi automata	120
5.2.2	LTL_{det} : Deterministic LTL	124
5.2.3	$LTL(F_s, G_s)$: LTL with only strict future/global operators . . .	125
5.3	Model checking $FO_{REG}(Reach + EGF)$	126
5.4	Several appetizer applications	127
5.4.1	Pushdown systems	127
5.4.2	Prefix-recognizable systems	128
5.4.3	Regular ground tree rewrite systems	129
5.5	A nonelementary lower bound for HM-logic	131
6	More applications of algorithmic metatheorems	135
6.1	PA-processes	135
6.2	Reversal-bounded counter systems	138
6.2.1	Basic model with one free counter	138
6.2.2	Extension with discrete clocks	142
6.3	Subclasses of Petri nets	144
6.3.1	Two-dimensional vector addition systems with states	145
6.3.2	Conflict-free Petri nets	146
6.3.3	Reversible Petri nets	147
II	Specific Approaches	149
7	Reversal-bounded counter systems and their extensions	151
7.1	Preliminaries	154
7.2	A Caratheodory-like theorem for linear sets	155
7.3	Parikh images of regular languages	160
7.3.1	A normal form theorem	160
7.3.2	Complementary lower bounds	164

7.4	Three simple applications	166
7.4.1	Integer programming	166
7.4.2	Decision problems for Parikh images of NWAs	168
7.4.3	Presburger-constrained graph reachability	173
7.5	Applications to model checking	175
7.5.1	LTL with complex fairness	175
7.5.2	Branching-time logics	177
8	One-counter processes	179
8.1	Preliminaries	180
8.2	Min-Max Arithmetic	182
8.2.1	The definition	182
8.2.2	Syntactic sugar: extended MMA	184
8.2.3	Basic properties of MMA and extended MMA	184
8.3	Saturations and small arithmetic progressions	187
8.3.1	Saturation construction	187
8.3.2	Computing small arithmetic progressions	188
8.3.3	Characterization of zero paths and positive paths	190
8.4	A translation to MMA	191
8.5	Application to weak-bisimilarity checking	196
8.6	Lower bounds	197
9	Networks of one-counter processes	201
9.1	Preliminaries	204
9.1.1	Asynchronous products	204
9.1.2	Synchronization predicates	205
9.2	Two fragments \mathfrak{L} and \mathfrak{L}' of Presburger Arithmetic	206
9.3	Complexity upper bounds	208
9.3.1	Combined and data complexity of $\text{FO}_{\mathfrak{S}}(\text{Reach})$	208
9.3.2	Expression complexity of $\text{FO}^2(\text{Reach})$	212
9.4	Complexity lower bounds	213
III	Epilogue	217
10	Conclusions and Future work	219

Bibliography	223
A Proofs from Chapter 4	245
A.1 Proposition 4.1.5 implies necessity in Lemma 4.1.4	245
A.2 Proof of Lemma 4.2.2	246
A.3 Proof of Proposition 4.3.4	247
A.4 Proof of Proposition 4.3.7	248
B Proofs from Chapter 5	249
B.1 Proof of Proposition 5.4.6	249
C Proofs from Chapter 7	251
C.1 Proof of Fact 7.3.3	251
C.2 Proof of Lemma 7.3.5	252
C.3 Proof of Lemma 7.3.6	252
C.4 Proof of Proposition 7.3.10	252
C.5 Proof of Proposition 7.4.2	254
C.6 Proof of Proposition 7.5.7	255
D Proofs from Chapter 8	257
D.1 Proof of Lemma 8.2.2	257
D.2 Proof of Lemma 8.3.1	261
D.3 Proof of Lemma 8.3.3	262
D.4 Proof of Theorem 8.4.3	262
D.5 Proof of Proposition 8.6.2	264
D.6 Proof of Proposition 8.6.3	269
E Proofs from Chapter 9	273
E.1 Proof of Proposition 9.2.1	273
E.2 Missing proofs from Subsection 9.3.2	274
E.3 Proof of Lemma 9.4.1	279
E.4 Proof sketch of Proposition 9.4.4	280
E.5 Proof of Proposition 9.4.5	280
E.6 Proof of Proposition 9.4.6	282

Chapter 1

Introduction

The past few decades saw an unprecedented growth rate of computers in scale and functionality. This has resulted in a substantial growth in complexity, which consequently increases the likelihood of subtle errors. It is a truism that in this technological era people have grown accustomed to systems that from time to time exhibit certain faults. Although such faults are a mere nuisance for everyday systems (e.g. personal desktops “hang”), they could be catastrophic for safety-critical or life-critical systems. Furthermore, it is well-known that, even when the systems are not safety-critical or life-critical, errors could still result in a substantial loss of money or productivity¹. Many examples of such system failures and their impacts are well-documented (e.g. see [Cip95, CGP99, Gre09]).

For a long time, testing has been the standard technique for system validation. Nowadays, testing is well-known to be insufficient to ensure the correctness of a system. This statement is even truer in the presence of concurrency in the system. To ensure that a system is correct, formal methods are necessary. Model checking is a *fully-automatic* formal verification method that has been extremely successful in validating and verifying safety-critical systems in the past three decades resulting in a recent bestowal of ACM Turing Award to its pioneers. Loosely speaking, in order to check that a system satisfies a certain property, we first create an *abstract model* \mathcal{S} (usually as a finite transition system) that captures how the system evolves, and express the property as a formula ϕ in some *logical language* (usually some temporal logic). This reduces the initial problem to checking whether \mathcal{S} satisfies ϕ , which can then be checked using standard model checking algorithms (e.g. see [CGP99, Sch02]).

¹The 80/20 rule is a well-known rule of thumb stating that only 20% of software development effort is spent on writing codes, while the rest is primarily spent on debugging

Model checking primarily differs from other approaches in the literature of verification (e.g. *automated theorem proving* and *traditional static analysis*) in two aspects. First of all, model checkers are meant to be *fully-automatic*, i.e., can be used as a blackbox. This is in contrast to automated theorem provers, which often require considerable user interventions. This aspect of model checking perhaps explains the wide adoption of model checking technologies by industries including NASA, Intel, IBM, and Motorola. The other major difference of model checking is the use of *expressive* specification language, e.g., temporal logics like LTL (Linear Temporal Logic), and CTL (Computation Tree Logic). This is in contrast to traditional static analysis techniques, which are fully-automatic but admit only very simple properties like safety and liveness; see [DKW08] for a more detailed discussion. Nowadays model checking is widely used for static analysis of programs (cf. [DKW08]).

In theory, real-world systems can almost always be modeled as finite transition systems that are *explicitly* represented (e.g. using adjacency lists). However, such a naive approach is often impractical. One well-known problem with this approach is the *state-explosion problem*, i.e., the number of configurations in the abstract model grows exponentially in the number of certain parameters in the actual system. For example, a distributed protocol with n processes could have at least exponentially many possible configurations. One successful approach to deal with this problem is called *symbolic model checking* [BCM⁺90, McM93], which is to develop model checking algorithms on *symbolic representations* of the transition system. In the case of [BCM⁺90, McM93], the symbolic representation is ordered binary decision diagrams (OBDDs). An intuitive explanation of the success of this approach is that many real world systems exhibit a large amount of symmetry and therefore could be *succinctly* represented as OBDDs, on which efficient algorithms could be developed.

In the past fifteen years, there has been a lot of work in extending the symbolic model checking techniques to deal with symbolic representations of *infinite-state* transition systems. Although most real-world systems could be thought of as finite systems (e.g. the size of hard disks and the number of processes of a distributed protocol are finite in reality), it is often more suitable to model them as infinite-state systems. For example, in the study of distributed algorithms [Lyn96], a distributed protocol is said to satisfy a certain property (e.g. freedom from deadlock) if *each* instance of the protocol with n processes satisfies the property, i.e., not only for each value of n up to (say) 1500, although this number could be reasonable for today's standard. This is arguably also the reason why abstract models of computation such as Turing machines (with

an infinite tape) and Minsky’s counter machines (with the ability to store unbounded integer values) are used as formal definitions of the intuitive notions of algorithms. We shall now mention a few possible sources of infinity in the *abstractions* of real-world systems:

1. Data structures: stacks (e.g. for modeling recursions), queues (e.g. for modeling communication channels), arrays and heaps.
2. Numeric data types: integers, reals, etc.
3. Discrete or real-valued clocks.
4. Concurrency: unbounded number of processes.

Most of these sources of infinity can easily result in Turing-powerful models of computation (in the sense of being able to generate the computation graphs of Turing machines up to isomorphisms). Despite this, researchers have obtained promising results in this direction that are both interesting from both practical and theoretical points of view.

Approaches to infinite-state model checking that have been considered in the literature can often be (somewhat loosely) classified into two categories: “generic” and “specific”. *Generic* approaches usually adopt powerful symbolic representations of infinite-state systems (i.e., those that can capture Turing-powerful models of computation such as Turing machines or counter machines) and develop partial techniques for solving model checking problems over such systems. These partial techniques might turn out to be complete (i.e. yield decidability) in cases when certain restrictions are imposed. In contrast, *specific* approaches avoid undecidability by always restricting to non-Turing-powerful formalisms *at the outset*. Nonetheless, this does *not* necessarily mean that positive results obtained in this way are always restrictive. In this thesis, we shall present generic techniques and specific techniques for obtaining decidability with optimal (or near-optimal) computational complexity of various infinite-state model checking problems.

The rest of this section is organized as follows. In Section 1.1 and Section 1.2, we shall review some results in the literature of infinite-state model checking that have been obtained using specific and generic approaches, respectively. In Section 1.3, we will discuss the contributions of this thesis. Finally, in Section 1.4 we will outline how the thesis is organized.

1.1 Specific approaches

Finding classes of infinite-state systems with decidable model checking tasks is by far the most popular approach in infinite-state model checking. This perhaps explains the plethora of decidability results that have been obtained in infinite-state model checking. We shall now review some of the major decidability/complexity results in the area.

One of the earliest decidability results in infinite-state model checking that permits an expressive specification language is arguably Muller and Schupp's result that model checking monadic second-order logic (MSO) over *pushdown systems* (i.e. the transition graphs of pushdown automata) is decidable [MS85]. Pushdown systems are relevant in verification since they are known to be good abstractions for sequential programs with unbounded recursions. On the other hand, there exists a fixed pushdown system (i.e. the infinite binary tree) with a nonelementary² complexity of MSO model checking [Sto74]. This is in contrast to the problem of reachability over pushdown systems, which is easily reducible to the P-complete problem of nonemptiness of languages of pushdown automata. This motivated researchers to find logics that are weaker than MSO, but are still sufficiently expressive for verification purposes, i.e., they should be able to express reachability and possibly also some liveness properties³. Temporal/modal logics turn out to have much better complexity over pushdown systems. Walukiewicz [Wal96, Wal01] was the first to identify that model checking (modal) μ -calculus over pushdown systems is EXP-complete. This specification language is subsumed by MSO, but turns out to be as powerful as MSO for expressing bisimulation-invariant properties [JW96], which include most properties of interests in verification. Linear Temporal Logic (LTL) was then proved to have EXP-complete model checking complexity over pushdown systems [BEM97]. In contrast to μ -calculus, model checking LTL was shown to be solvable in P for a fixed formula, which is appealing since LTL specifications are quite small in practice. The complexity for other temporal logics including CTL (Computation Tree Logic), EF-logic, and Propositional Dynamic Logic (PDL) have also been identified to be within EXP (cf. [BEM97, GL06, Wal00]).

Many of the results for pushdown systems have by now been extended to more expressive classes of infinite-state systems, which we shall mention next. First, the decidability of MSO has been extended by Caucal [Cau96, Cau03] to *prefix-recognizable*

²This means that the time complexity cannot be bounded from above by k -fold exponential functions for every fixed $k > 1$.

³In this sense, *Hennesy-Miler logic* (or modal logics of actions) does not fall within this category

systems, which can be understood as pushdown systems with infinitely many rewrite rules compactly represented by means of regular languages. The complexity of modal and temporal logics have also been identified for prefix-recognizable systems [Cac02, GL06, KPV02]. For example, over μ -calculus the problem remain EXP-complete [Cac02, KPV02]. Over LTL, the problem remains EXP-complete, but the EXP lower bound still holds for a fixed formula [KPV02]. Interestingly, it was only shown recently [Göl08] that reachability over prefix-recognizable systems is already EXP-hard. *Caucal* [Cau02] also gave another extension of the MSO decidability of prefix-recognizable systems to a hierarchy of infinite graphs, which are known as *Caucal hierarchy*. As has been shown in [CW03], this hierarchy of graphs is intimately connected to a formalism called *higher-order pushdown automata* [Mas76], which extend pushdown automata by “stack-of-stacks” structures. Some results on model checking higher-order pushdown systems are also known, e.g., model checking μ -calculus is n -EXP complete for order- n higher-order pushdown systems [Cac03, CW07]. Similar results on related formalisms like higher-order recursion schemes and collapsible higher-order pushdown automata, which are suitable abstractions for higher-order programs with unbounded recursions, are also known (cf. [Ong06, HMOS08]).

So far, we have only discussed abstract models of sequential programs. We now discuss models of concurrent programs. *Petri nets* — initially proposed by Carl Adam Petri — are one of the first well-known models for *purely* concurrent programs with interesting decidability results. Roughly speaking, they are a subclass of Minsky’s counter machines with only one state that cannot test whether a counter value is zero. Reachability for Petri nets is known to be solvable in non-primitive recursive time [May84], but is only known to be EXPSPACE-hard [Lip76]. On the other hand, branching-time model checking over Petri nets is known to be undecidable [Esp97a] even over EF-logic, which is probably the simplest standard branching-time logic with a reachability operator. Despite this, LTL model checking is decidable, but has been shown to be as hard as reachability for Petri nets [Esp94]. Interestingly, when only infinite runs are considered, the complexity of the problem is EXPSPACE-complete [Hab97]. Many subclasses of Petri nets with better decidability/complexity are known. We shall mention *communication-free Petri nets* (a.k.a. *basic parallel processes*), which are simply Petri nets whose transitions depend only on the value of a single counter. Communication-free Petri nets are known to have NP-complete reachability problem [Esp97b] and PSPACE-complete EF-logic model checking problem [May98]. We refer the reader to the survey [Esp96] and the thesis [May98] for more results and

discussions on Petri nets and their subclasses.

The expressive power of Petri nets and pushdown automata as graph generators are known to be incomparable up to bisimulation (cf. [BCMS01, Mol96]). Intuitively, this is because pushdown systems can only model sequential programs, while Petri nets only purely concurrent programs. Some research has been made into combining them to obtain a model that is both sequential and concurrent. We shall first mention *PA-processes* (cf. [BW90, BCMS01, Mol96]), which are obtained by blending one-state pushdown automata and communication-free Petri nets. PA-processes are known to have decidable EF-logic model checking and NP-complete reachability [May98, LS02], but undecidable LTL model checking [BH96]. It is also known that they can be used to model parallel programs with unbounded recursions and unbounded parallelism [BW90, EP00]. The first complete generalization of pushdown automata and Petri nets was given by Mayr [May98], which he calls *Process Rewrite Systems (PRS)*. Despite its generality, PRS is still known to have decidable reachability problem [May98].

Another way of incorporating some concurrency into pushdown systems is to consider rewrite rules over ranked trees instead of words. This approach yields a class of infinite-state systems that is called *ground tree rewrite systems* (cf. [CDG⁺07]). It has been shown that ground tree rewrite systems have polynomial-time reachability [CDGV94, Löd03] and decidable model checking with respect to first-order logic with reachability operators [DT90]. Löding [Löd03] was the first to show that some interesting liveness properties could be also decided for ground tree rewrite systems. In particular, he showed that the problem of checking the existence of an infinite path from a given tree T which visits a given set $\mathcal{L}(\mathcal{A})$ infinitely often, where $\mathcal{L}(\mathcal{A})$ is the language of a tree automaton \mathcal{A} , is decidable in polynomial time. Such a liveness property is often referred to as *recurrent reachability* and *repeated reachability* (e.g. see [BBF⁺01]). In order to emphasize the expressive “target” set $\mathcal{L}(\mathcal{A})$, we shall also address the property as recurrent reachability with *regular fairness constraint*. These positive results also extend to *regular ground tree rewrite systems* [DT90, Löd03, Löd06], which are extensions of ground tree rewrite systems with infinitely many rules compactly represented by means of tree automata (i.e. similar to prefix-recognizable systems). Despite this, it can be shown that model checking logics like LTL and CTL is undecidable over ground tree rewrite systems.

So far, we have discussed some results on systems with two sources of infinity, i.e., stacks (or generalizations thereof) and concurrency. What about numeric data types

like integers? Unfortunately, adding numeric data types easily result in undecidability, e.g., consider Minsky's 2-counter machines. As we saw earlier, decidability can be retained if we do not allow zero tests yielding the model called Petri nets. However, this is not satisfactory since programs naturally perform arithmetic expressions, the simplest of which already require zero test. Let us now discuss some restrictions on counter machines that still allow test for zero but still have some interesting decidability results. Firstly, if we restrict the number of counters to one, we obtain 1-counter machines, which can be thought of as pushdown automata with one stack symbol plus a non-removable stack-bottom symbol. In this way, 1-counter systems inherit the decidability results from pushdown systems, e.g., model checking MSO. It turns out, though, that 1-counter systems have better computational complexity. For example, LTL and μ -calculus model checking over 1-counter systems were shown to be PSPACE-complete [Dem06, Ser06], in contrast to pushdown systems which are EXP-complete. For EF-logic, the complexity is known to be in PSPACE [Wal00] and DP-hard [JKMS04]. Another well-known decidable restrictions of counter machines are *reversal-bounded counter machines*, which were initially proposed by Ibarra [Iba78]. These are simply counter machines each of whose counters can change from a non-decreasing mode to a non-increasing mode (or vice versa) for a fixed r number of times. Reachability was initially shown by Gurari and Ibarra [GI81] to be solvable in PSPACE, and later was shown in [HR87] to be precisely NP-complete when r is given in unary and NEXP-complete when r is given in binary. Furthermore, when the number of reversals and the number of counters are fixed in advance, the problem is solvable in polynomial time [GI81]. Certain liveness problems like recurrent reachability have also been shown to be decidable [DIP01] for reversal-bounded counter systems with one free counter.

Results that combine infinite data structures with numeric data types are also available. We shall only mention the result on reversal-bounded counter systems with a pushdown stack and finitely many discrete clocks [DIB⁺00]. This class of systems generalizes pushdown systems, reversal-bounded counter systems, and discrete timed systems [AD94] simultaneously. Despite this, it was shown in [DIB⁺00] that interesting safety properties are still decidable. It was an open question in [DIP01] whether interesting liveness properties are also decidable for this model.

There are also other classes of systems with interesting decidability results for model checking that we have not mentioned. These include lossy channel systems [ACJT96] and probabilistic infinite-state systems including probabilistic pushdown systems (cf. [KEM06]), which we will not further encounter in the thesis.

1.2 Generic approaches

We have hitherto mentioned only models of computations that are *not* Turing-powerful. However, these are not the only models that were intensively studied in infinite-state model checking communities. Many formalisms that are capable of generating the configuration graphs of Turing machines or Minsky’s counter machines have also been studied. These include rational transition systems [Mor00, BG09], automatic and ω -automatic structures [Blu99, BG04], tree-automatic structures [Blu99, BLN07], and Presburger-definable systems (cf. [Boi99, BFLP08, FL02]). Since even reachability is already undecidable over such systems, most results concerning verification over such systems have a semi-algorithmic flavor. In particular, we mention the work on *regular model checking*, which aims to develop practical semi-algorithmic techniques for computing a symbolic representation (e.g. using regular languages) of the reachability sets or reachability relations of such systems. The reader is referred to [AJNS04, Bou01, Boi99, BLW03, BJNT00, BHV04, KMM⁺01, Nil05] for more details. Many of the semi-algorithms given in this literature, however, do not come with a completeness criterion, i.e., a criterion on the input systems whereby the semi-algorithms will certainly terminate with a correct answer. In other words, the performance of many of these semi-algorithms is only evaluated experimentally. In the case when completeness criteria are given, they are often unnatural and do not subsume commonly considered *subclasses* of systems with decidable model checking problems.

Recently, there have been several successful attempts to provide semi-algorithms with natural criteria for completeness. In particular, we shall mention the work [LS05a, BFLS05], which provide semi-algorithms based on the “acceleration techniques” of [CJ98, FL02, Boi03] for computing symbolic representations of reachability sets or reachability relations over linear counter systems (a subset of Presburger-definable systems). They show that their algorithms terminate with a correct answer iff the input systems are *flattable*, i.e., they can be turned into a *flat* counter system [CJ98]. Many interesting subclasses of counter systems have been shown to satisfy this property, e.g., 2-dimensional vector addition systems with states [LS04], reversal-bounded counter systems [LS05a], and other subclasses of Petri nets [LS05a]. Thus, this approach yields a *single* semi-algorithm that is guaranteed to solve the reachability problems for these subclasses of counter systems, instead of one dedicated algorithm for each subclass. Furthermore, the general procedures turn out to be simpler than the specialized algorithms (e.g. reachability for 2-dim vector addition systems was shown to be de-

cidable in [HP79] with a rather difficult technique). These semi-algorithms have also been implemented in FAST [BFLP08] with impressive experimental results.

The results of [LS05a, BFLS05] can naturally be viewed as *algorithmic metatheorems*, as was suggested by the authors. More precisely, to prove whether a subclass of linear counter systems has decidable reachability, it suffices to show that they are flattable. In this sense, other results in the verification literature can also be classified as algorithmic metatheorems. In particular, we shall mention the works of [Fin87, Fin90, ACJT96, FS01] on *well-structured transition systems* and the works of [Sem84, Wal02, CW98] on operations on transition systems that preserve decidability of monadic second-order logic. In the case of finite-state model checking, algorithmic metatheorems are also used extensively to obtain good algorithmic bounds for evaluating logical formulas [FG06].

1.3 Contributions

The main contributions of this thesis are new generic and specific techniques for infinite-state model checking.

Our generic approach to infinite-state model checking adopts word/tree automatic transition systems [BG04] as generic frameworks. Although reachability is already undecidable, word/tree automatic transition systems are known to satisfy some nice closure/algorithmic properties, e.g., closure under boolean combinations and automata projections [Hod83]. Using these properties, we will prove various algorithmic metatheorems for showing decidability of model checking over word/tree automatic transition systems with optimal (or near-optimal) complexity. More importantly, we will show that many previously known or unknown decidability/complexity can be obtained in a *uniform* way using our metatheorems.

Thus far only algorithmic metatheorems for safety properties are available in the literature [LS05a, BFLS05]. We complement these results by providing algorithmic metatheorems for liveness. Our most basic algorithmic metatheorem concerns a particular liveness property over the expressive class of word/tree automatic systems called *recurrent reachability checking with regular fairness constraints*. Such a property is important since certain logic model checking (e.g. LTL) can be reduced to it. In particular, we show that, for any subclass C of word/tree automatic systems for which there exists an algorithm \mathcal{M} computing a word/tree automatic presentation of the reachability relation of a given system in C , we may decide recurrent reachability by first

computing the reachability relation of the given system and then perform some extra polynomial-time computation. This metatheorem will then be extended to recurrent reachability with *multiple* regular fairness constraints (a.k.a. *generalized Büchi conditions*). Roughly speaking, this problem asks whether there exists an infinite path from a given configuration (i.e. a finite word or a finite tree) visiting each of the given regular sets $\mathcal{L}(\mathcal{A}_1), \dots, \mathcal{L}(\mathcal{A}_n)$ infinitely often (simultaneous visits are not required). Together with the known results on algorithms for computing reachability relations for specific classes of infinite-state systems, our metatheorems can be applied to uniformly derive decidability of recurrent reachability over pushdown systems, prefix-recognizable systems, (regular) ground-tree rewrite systems, PA-processes, order-2 collapsible pushdown systems, PA-processes, reversal-bounded counter systems (and extensions thereof), and some subclasses of Petri nets including 2-dim vector addition systems. For many of these classes of infinite-state systems, we manage to obtain optimal complexity. Many of these decidability/complexity results were not previously known in the literature. For example, Löding [Löd06] asked whether his result on the decidability of recurrent reachability with a single regular fairness constraint over ground tree rewrite systems could be extended to multiple regular fairness constraints, which we answer positively using the techniques in this thesis.

Building on our algorithmic metatheorems for recurrent reachability, we provide algorithmic metatheorems for *logic model checking* over word/tree automatic systems. In particular, we consider the LTL (or fragments thereof) model checking problems with multiple regular fairness constraints. Fairness constraints are standard ways of eliminating executions that do not represent actual paths in the real-world systems, i.e., “spurious” executions that are introduced by abstractions (cf. [BBF⁺01]). Regular languages give powerful ways of expressing fairness, which cannot be expressed in LTL alone. Our results are as follows. To begin with, we show that if we additionally require the subclass \mathcal{C} of word/tree automatic systems to be *closed under products with finite systems*, then we obtain decidability of the full LTL model checking with multiple regular fairness constraints. We will use this algorithmic metatheorem for uniformly deriving decidability with optimal (or near-optimal complexity) of LTL model checking with multiple regular fairness constraints over pushdown systems, prefix-recognizable systems, and extensions of reversal-bounded counter systems with discrete clocks and one free counter. The condition of closure under products with finite systems turns out to be rather restrictive. For this reason, we provide a weakening of this condition, which we call *closure under taking subsystems*. This condition

is rather weak and is satisfied by virtually every class of infinite-state systems. Our second algorithmic metatheorem is that if \mathcal{C} satisfies this condition together with the condition for recurrent reachability, we have decidability of the fragments $\text{LTL}(\mathbf{F}_s, \mathbf{G}_s)$ and LTL_{det} over \mathcal{C} with multiple regular fairness constraints. These fragments of LTL are sufficiently powerful to express many interesting safety/liveness constraints. More importantly, we use this algorithmic metatheorem to give new decidability/complexity results over many of the classes of infinite-state systems.

Additionally, we build on top of our algorithmic metatheorems for recurrent reachability to obtain algorithmic metatheorems for extensions of first-order logic with reachability and recurrent reachability operators (possibly enriched with path constraints). Similarly, we can apply these metatheorems for deriving decidability results over classes of infinite-state systems that have been considered in the literature.

Generic approaches are not without limitations. In particular, when we consider subclasses of counter systems (e.g. one-counter systems and reversal-bounded counter systems), our generic approaches cannot immediately derive optimal complexity. We address this problem by providing new techniques that are designed specifically for these classes of systems.

First of all, we will develop techniques to compute Parikh images of nondeterministic finite state automata (as semilinear sets) in a more efficient way. This technique can then be used to derive an optimal complexity for LTL model checking with multiple regular fairness constraints over reversal-bounded counter systems with discrete clocks. We will also provide a kind of fixed-parameter tractability result for model checking EF-logic over reversal-bounded counter systems. These results were not previously known.

Finally, we will consider the problem of model checking EF-logic and first-order logic with reachability over one-counter processes and networks of one-counter processes with no bounds on the number of reversals. As we will show later, these classes of systems form a natural subclass of programs with multiple integer variables and simple synchronizations between the variables. To prove optimal complexity for these model checking problems, we will first introduce new subclasses of Presburger arithmetic and prove that they have good complexity (all below PSPACE). The optimal complexities of the model checking problems are then derived by polynomial reductions to the membership problems for these subclasses of Presburger arithmetic.

1.4 Organizations

The thesis is organized as follows. We shall first recall necessary preliminaries in Chapter 2. The contents after Chapter 2 are divided into three parts:

1. Part I contains generic techniques for infinite-state model checking. In particular, we shall review basic results on word/tree automatic transition systems in Chapter 3. In Chapter 4, we will prove algorithmic metatheorems for recurrent reachability properties and their extensions over word/tree automatic systems. In Chapter 5, we will extend the algorithms from Chapter 4 to logic model checking. We will use these metatheorems in the corresponding chapter to derive some known or previously unknown decidability/complexity results in infinite-state model checking. In Chapter 6, we will study more applications of our algorithmic metatheorems.
2. Part II contains specific techniques. In particular, we will deal with model checking problems over reversal-bounded counter systems and their extensions in Chapter 7. In Chapter 8, we will study model checking problems over one-counter processes. Finally, we will consider model checking problems over networks of one-counter processes in Chapter 9.
3. Part III contains a summary of the results in the thesis and future work.

As a convention, we shall use ■ to end remarks, and ♣ (resp. □) to end examples (resp. proofs).

Chapter 2

Preliminaries

In this chapter, we shall fix some notations that will be used in the sequel, and review basic definitions and results from automata theory, complexity theory, and logic. The reader is assumed to have basic familiarity with these subjects. This chapter is organized as follows. In Section 2.1, we fix some general mathematical notations that we shall use throughout the thesis. Automata theory is perhaps the most important tool in the thesis. We shall review necessary preliminaries from automata theory in Section 2.2. In Section 2.3, we review standard definitions and results from computability and complexity theory. Most mathematical structures that we will encounter in the sequel can be formalized as transition systems or logical structures over some vocabularies. We shall review them in Section 2.4. Finally, Section 2.5 reviews the logics and properties that we will deal with in the sequel.

2.1 General notations

Most mathematical notations and terminologies that we use in this thesis are fairly standard. For the sake of completeness, we shall mention some of these in this section.

Some notations from set theory We use standard notations for set operations: union (\cup), intersection (\cap), set difference (\setminus), Cartesian product (\times), and power set (e.g. 2^S for a given set S). Given n sets S_1, \dots, S_n , their product $\prod_{i=1}^n S_i$ is the set $\{(s_1, \dots, s_n) : \forall i \in [1, n](s_i \in S_i)\}$. If $S_1 = \dots = S_n$, then this set is also written S_1^n . Denote by ω the least infinite ordinal.

Sets of numbers and vector spaces Let \mathbb{N} be the set of nonnegative integers. As usual, we use \mathbb{R} , and \mathbb{Z} to denote, respectively, the set of real numbers, and the set of integers. We also often use such notations as $\mathbb{R}_{\geq 0}$ and $\mathbb{Z}_{>0}$, which in this case mean the set of nonnegative real numbers and the set of positive integers, respectively. Given two integers $i < j$, we use interval notations of the form $[i, j]$ to denote the set $\{i, i+1, \dots, j\}$ of integers (instead of reals), which is more standard in computer science. Similarly, we shall use notations like $(i, j]$ to mean the set $[i, j]$ but excluding extreme points (in this case i). Logarithm notations used in this thesis has base 2. When we fix some vector space \mathbb{R}^k , we use $\mathbf{0}$ to denote the element $(0, \dots, 0)$ in \mathbb{R}^k . We shall also denote by $\{\mathbf{e}_i\}_{i=1}^k$ the standard basis for \mathbb{R}^k , where \mathbf{e}_i denotes the vector with all-zero entries except for the i th.

Partial orders Recall that a partial order \preceq on a set S is *well-founded* if there does not exist a strictly decreasing infinite sequence $s_1 \succ s_2 \succ \dots$ of elements from S . An element s of S is said to be \preceq -minimal, if all $s' \in S$ with $s' \preceq s$ satisfies $s = s'$.

In the sequel, we shall reserve \preceq for the component-wise partial order on \mathbb{N}^k , i.e., $(a_1, \dots, a_k) \preceq (b_1, \dots, b_k)$ iff $a_i \leq b_i$ for all $i \in \{1, \dots, k\}$. Dickson's lemma [Dic13] states that \preceq is well-founded.

Asymptotic notations We use the following standard asymptotic notations, especially when measuring the computational complexity of a problem: big-oh $O()$, small-oh $o()$, and big omega $\Omega()$.

Arithmetic on $2^{\mathbb{Z}^k}$ First, we extend standard arithmetic operations (addition, subtraction, and multiplication) to tuples in a component-wise manner. These can be further extended to sets of tuples as follows. Given two sets $S_1, S_2 \subseteq \mathbb{Z}^k$, we define the operation $\odot \in \{+, -, \cdot\}$ on them as follows: $S_1 \odot S_2 := \{\mathbf{v}_1 \odot \mathbf{v}_2 : \mathbf{v}_1 \in S_1, \mathbf{v}_2 \in S_2\}$. For $n \in \mathbb{N}$ and $S_2 \subseteq \mathbb{N}$, we shall also write $n \odot S_2$ to mean $\{n\} \odot S_2$. An *arithmetic progression* is any set of numbers of the form $a + b \cdot \mathbb{N}$ for some $a, b \in \mathbb{N}$. The number a (resp. b) is said to be the *offset* (resp. the *period*) of $a + b\mathbb{N}$.

2.2 Automata theory

In this section, we shall review some basic definitions and results from automata theory. In particular, we will look at finite-state automata on finite words, finite trees, ω -words,

and ω -trees. We will also briefly recall pushdown automata and context-free grammars used as generators of languages of finite words. Finally, we will look at Parikh's Theorem, which relates the sets of letter-counts of languages recognized by regular languages and context-free languages (on finite words) and semilinear sets. For a more thorough treatment of the subject, the reader is referred to [Koz97, Sip97, Tho96].

2.2.1 Automata over finite words

Languages over finite words

An *alphabet* Σ is simply a finite nonempty set of *letters*. We say that Σ is k -ary if $|\Sigma| = k$. A *word* (or *string*) over Σ is a finite sequence $w = a_1 \dots a_n$ where $a_i \in \Sigma$ for each $1 \leq i \leq n$ and $n \in \mathbb{N}$. If $n = 0$, then w is the unique *empty word* ε . For $1 \leq i \leq j \leq n$, we write $w[i, j]$ to refer to the word $a_i a_{i+1} \dots a_j$. The word $w[i, j]$ is a *subword* of w . Note also that $w[i, i]$ refers to the i th letter a_i in w . For convenience, we shall also use $w[i]$ for $w[i, i]$. Given two words $u = a_1 \dots a_n$ and $v = b_1 \dots b_m$, the *concatenation* $u.v$ of u and v is the new word $a_1 \dots a_{n+m}$, where $a_{n+i} := b_i$ for each $1 \leq i \leq m$ (e.g. the concatenation of aba with bbb is $ababbb$). Note that ε is the unique word satisfying $\varepsilon.u = u.\varepsilon = u$ for each word u over Σ . For convenience, we shall often write uv instead of $u.v$ in the sequel. Given a number $n \in \mathbb{N}$, we define w^n as the concatenation of w with itself n times (e.g. for $w = ab$, we have $w^0 = \varepsilon$, $w^1 = w$, $w^2 = abab$). A word $w = a_1 \dots a_n$ has *length* $|w| = n$. Note that $|\varepsilon| = 0$. For a given letter $a \in \Sigma$, we denote by $|w|_a$ the number of occurrences of the letter a in w (e.g. $|aaba|_a = 3$). We denote by Σ^* (resp. Σ^+) the set of all words (resp. nonempty words) over Σ . In the sequel, when we omit mention of Σ when referring to these notions, we tacitly assume some underlying alphabet Σ .

A *language* (over Σ) is any subset $\mathcal{L} \subseteq \Sigma^*$. We shall define a number of useful operations on languages. Standard set operations such as union (\cup), intersection (\cap), and complement (\setminus) — also known as *boolean operations* — can be applied to languages as usual. For a language \mathcal{L} , we use $\overline{\mathcal{L}}$ to denote the complement $\Sigma^* \setminus \mathcal{L}$ of \mathcal{L} . Given two languages \mathcal{L} and \mathcal{L}' over Σ , we define their *concatenation*

$$\mathcal{L}.\mathcal{L}' := \{uv : u \in \mathcal{L}, v \in \mathcal{L}'\}.$$

As before, we will mostly use the notation $\mathcal{L}\mathcal{L}'$ instead of $\mathcal{L}.\mathcal{L}'$. For each $n \in \mathbb{N}$, we

define \mathcal{L}^n and $\mathcal{L}^{\leq n}$ to be the languages defined as follows

$$\begin{aligned}\mathcal{L}^n &:= \{u_1 \dots u_n : u_1, \dots, u_n \in \mathcal{L}\} \\ \mathcal{L}^{\leq n} &:= \bigcup_{i=0}^n \mathcal{L}^i.\end{aligned}$$

Finally, we define the *Kleene star* of \mathcal{L} to be the language

$$\mathcal{L}^* := \bigcup_{i \in \mathbb{N}} \mathcal{L}^i.$$

Regular languages and regular expressions

We now recall the notion of regular languages, along with regular expressions as their standard finite representations. We begin by recalling the syntax of *regular expressions* e over an alphabet Σ using the standard Backus-Naur Form:

$$e, e' ::= \varepsilon \mid a \ (a \in \Sigma) \mid e + e' \mid e.e' \mid e^*.$$

The three operators here are union (+), concatenation (.), and Kleene star (*). The language $\mathcal{L}(e)$ *generated* by a regular expression e can be defined by induction:

- $\mathcal{L}(\varepsilon) := \varepsilon$.
- $\mathcal{L}(a) := \{a\}$ for each $a \in \Sigma$.
- $\mathcal{L}(e + e') := \mathcal{L}(e) \cup \mathcal{L}(e')$.
- $\mathcal{L}(e.e') := \mathcal{L}(e).\mathcal{L}(e')$.
- $\mathcal{L}(e^*) := \mathcal{L}(e)^*$.

Let us now define some syntactic sugar. We shall allow the expression Σ with the obvious meaning $\mathcal{L}(\Sigma) = \Sigma$. When the meaning is clear, we shall also write ee' instead of $e.e'$ for two given regular expressions e and e' . An example of a regular expression is $(ab)^* + a^*$, which describes the set of all words that are of the form $(ab)^n$ or a^n for some $n \in \mathbb{N}$. To minimize the use of brackets ‘(’ and ‘)’, we assign an operator precedence in the following order (highest to lowest): ‘*’, ‘.’, and then ‘+’. Furthermore, observe that both of the operators ‘+’ and ‘.’ are associative: $\mathcal{L}(e + (e' + e'')) = \mathcal{L}((e + e') + e'')$ and $\mathcal{L}(e.(e'.e'')) = \mathcal{L}((e.e').e'')$. Therefore, we will write $(ab)^* + b^* + a^*$ instead of $((ab)^* + b^*) + a^*$. *Regular languages* (over Σ) are those languages that are generated by some regular expressions (over Σ). We now state a standard result about regular languages (e.g. see [Koz97] for a proof).

Proposition 2.2.1 *Regular languages are effectively closed under union, intersection, complementation, composition, and Kleene star.*

Finite automata

We now recall the notions of finite automata. A *nondeterministic word automaton* (NWA) over an alphabet Σ is a tuple $\mathcal{A} = (\Sigma, Q, \delta, Q_0, F)$ where

- Q is a finite set of *states*,
- $Q_0 \subseteq Q$ is a set of *initial* states,
- $F \subseteq Q$ is a set of *final* states, and
- $\delta \subseteq Q \times \Sigma \times Q$ is a *transition relation*.

We shall use $\mathbf{States}(\mathcal{A})$ to denote the set Q of states of \mathcal{A} . As usual, we shall also treat the transition relation δ as a transition function $\delta_f : Q \times \Sigma \rightarrow 2^Q$ such that $q' \in \delta_f(q, a)$ iff $(q, a, q') \in \delta$. If $|Q_0| = 1$ and $|\delta(q, a)| \leq 1$ for each $q \in Q$ and $a \in \Sigma$, then we say that \mathcal{A} is *deterministic*. In this case, \mathcal{A} is a DWA (deterministic word automaton). A *path* π in \mathcal{A} from $q \in Q$ to $q' \in Q$ is simply an interleaving sequence $p_0 a_1 p_1 \dots a_m p_m$ of states in Q and letters in Σ such that $p_{i+1} = \delta(p_i, a_{i+1})$ for each $i \in [0, m)$. It is said to be a *run* if $p_0 \in Q_0$. We shall also use $\mathcal{L}(\pi)$ to denote the *path labels* $a_1 \dots a_m$, and say that π is a *path on (input) $a_1 \dots a_m$* . For convenience, we shall sometimes omit the path labels from π , and simply refer to it as a path $\pi = p_0 \dots p_m$ on the word $a_1 \dots a_m$. In addition, the path π has length $|\pi| = m$, and we let $\mathbf{first}(\pi) := p_0$ (resp. $\mathbf{last}(\pi) := p_m$) denote the initial (resp. end) state in the path π . For $1 \leq i \leq j \leq m$, we shall use the notation $\pi[i, j]$ (resp. $\pi(i)$) to denote the path $p_i a_{i+1} p_{i+1} \dots a_j p_j$ (resp. node p_i). Given two paths $\pi = p_0 a_1 \dots p_m$ and $\pi' = p_m a_{m+1} \dots p_n$ (with $m \leq n$), we let $\pi \odot \pi'$ denote the concatenated path $p_0 a_1 \dots p_m a_{m+1} \dots p_n$. A path that ends in some final state $q \in F$ is said to be *accepting*. The NWA \mathcal{A} is said to *accept* the word $w \in \Sigma^*$ from q if there exists an accepting path of \mathcal{A} on w from q . When we say \mathcal{A} accepts the word w without mention of the state q , we tacitly assume that q is the initial state of \mathcal{A} . The language $\mathcal{L}(\mathcal{A})$ *accepted* by \mathcal{A} is simply the set of words over Σ accepted by \mathcal{A} . As usual, for clarity we may define a finite automaton by drawing an edge-labeled directed graph whose nodes (resp. arcs) define the states (resp. transitions) of the automaton. In the sequel, we use filled circles to denote final states, while the initial state is defined by drawing a source-less incoming arc to a node. For example, the language $\mathcal{L}((ab)^* + a^*)$ is accepted by the automaton in Figure 2.1.

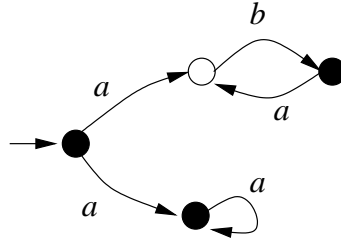


Figure 2.1: An NFA recognizing the language $\mathcal{L}((ab)^* + a^*)$.

Remark 2.2.1 Recall that our definition of NWAs allow more than one initial states. This is done only for convenience since we may easily construct an equivalent NFA with only one initial state by adding one extra state. In the sequel, we shall often assume that an NFA has only one state q_0 and write $(\Sigma, Q, \delta, q_0, F)$ instead of $(\Sigma, Q, \delta, \{q_0\}, F)$. ■

Complexity measure

For the purpose of complexity analysis, we shall define complexity measures for automata and regular expressions.

We start with regular expressions. The size $\|e\|$ of a regular expression can be defined inductively as follows:

1. for each $a \in \Sigma$, $\|a\| := 1$,
2. $\|e + e'\| := \|e\| + \|e'\| + 1$,
3. $\|e.e'\| := \|e\| + \|e'\| + 1$, and
4. $\|e^*\| := \|e\| + 1$.

In other words, $\|e\|$ is the number of nodes in the parse tree of e . Observe that the number of bits needed to write an expression e is at most $O(\|e\| \times \log |\Sigma|)$.

We now define computational complexity measures for NWAs. Given an NFA $\mathcal{A} = (\Sigma, Q, \delta, Q_0, F)$, the easiest, but less precise, measure can be obtained by measuring the number $|Q|$ of states and the size $|\Sigma|$ of the alphabet separately. Such a measure is reasonable since all other parameters of \mathcal{A} are polynomial in $|Q|$ and $|\Sigma|$, e.g., $|\delta| \leq |Q|^2 \times |\Sigma|$. On the other hand, when a more accurate analysis is desired, we shall instead use the following measure. Let $\|\mathcal{A}\|$ be the number of pairs (q, q') such that $(q, a, q') \in \delta$ for some $a \in \Sigma$. In other words, $\|\mathcal{A}\|$ denotes the number of different

unlabeled transitions in δ . Then, assuming that each state in Q occurs in δ at least once (by easily removing isolated states in linear time), each parameter in \mathcal{A} can be expressed *linearly* in $\|\mathcal{A}\|$ and $|\Sigma|$. For example, the number $|Q|$ of states is at most $\|\mathcal{A}\|$, and the number of transitions in $|\delta|$ is at most $\|\mathcal{A}\| \times |\Sigma|$. Moreover, the number of bits needed to write down the automaton is at most $O(\|\mathcal{A}\| \log \|\mathcal{A}\| \times |\Sigma| \log |\Sigma|)$. In the sequel, we call $\|\mathcal{A}\|$ the (*unlabeled transition*) *size* of the NWA \mathcal{A} . When we intend to measure the number of states as the complexity measure, we shall be explicit about this.

Some basic results

We now state several basic results from automata theory over finite words. We first start with a standard result concerning the equivalence of regular expressions and finite automata (e.g. see [Koz97] for a proof).

Proposition 2.2.2 *Given a language \mathcal{L} over Σ , the following statements are effectively equivalent:*

- (1) \mathcal{L} is generated by a regular expression.
- (2) \mathcal{L} is accepted by an NWA.
- (3) \mathcal{L} is accepted by a DWA.

Furthermore, there is a linear-time translation from (1) to (2).

All the translations in the proposition above run in time *at most* exponential in the size of the input (e.g. see [Koz97]). It turns out that *every* translation from (2) to (3) could be exponential in the worst case even over unary alphabet [Chr86]. In particular, there exists a class $\{\mathcal{A}_n\}_{n \in \mathbb{Z}_{>0}}$ of NWAs \mathcal{A}_n with n states over the alphabet $\{a\}$ whose smallest equivalent DWA require at least $2^{\Omega(\sqrt{n \log n})}$ states. When the alphabet contains at least two letters, the lower bound can be improved to 2^n (e.g. see [Var95]). In addition, the lower bound of $2^{\Omega(\sqrt{n \log n})}$ holds also for translations from (1) to (3) even over unary alphabet since NWAs and regular expressions are polynomially equivalent over unary alphabet [Chr86, Mar02, To09b]. Furthermore, there also exists an exponential lower bound for translations from (3) to (1) even over alphabet of size four [GN08].

In the sequel, we will meet several complex constructions over NWAs, many of which can be understood in terms of simpler constructions over NWAs such as boolean operations. Therefore, we next state basic results on computing NWAs recognizing

boolean combinations of languages of the given NWAs (see [Koz97, Var95] for more details).

Proposition 2.2.3 *Given NWAs \mathcal{A} and \mathcal{B} over the alphabet Σ :*

- (1) *we can compute in time $O(|\Sigma| \times (\|\mathcal{A}\| + \|\mathcal{B}\|))$ an NWA of size $\|\mathcal{A}\| + \|\mathcal{B}\|$ recognizing the language $\mathcal{L}(\mathcal{A}) \cup \mathcal{L}(\mathcal{B})$,*
- (2) *we can compute in time $O(|\Sigma| \times (\|\mathcal{A}\| \times \|\mathcal{B}\|))$ an NWA of size $\|\mathcal{A}\| \times \|\mathcal{B}\|$ recognizing the language $\mathcal{L}(\mathcal{A}) \cap \mathcal{L}(\mathcal{B})$, and*
- (3) *if $n = |\mathbf{States}(\mathcal{A})|$, we can compute in exponential time a DWA with 2^n states recognizing the language $\overline{\mathcal{L}(\mathcal{A})}$.*

Proof. We shall describe only the first and the second constructions. The third is done by the standard subset construction (e.g. see [Koz97, Var95]), which we will not encounter in this thesis. Therefore, suppose that $\mathcal{A} = (\Sigma, Q^{\mathcal{A}}, \delta^{\mathcal{A}}, Q_0^{\mathcal{A}}, F^{\mathcal{A}})$ and $\mathcal{B} = (\Sigma, Q^{\mathcal{B}}, \delta^{\mathcal{B}}, Q_0^{\mathcal{B}}, F^{\mathcal{B}})$. Without loss of generality, we assume that $Q^{\mathcal{A}} \cap Q^{\mathcal{B}} = \emptyset$.

Let us start with the construction for (1). Define the NWA $\mathcal{T} = (\Sigma, Q, \delta, Q_0, F)$ as follows:

- $Q := Q^{\mathcal{A}} \cup Q^{\mathcal{B}}$.
- $Q_0 := Q_0^{\mathcal{A}} \cup Q_0^{\mathcal{B}}$.
- $\delta := \delta^{\mathcal{A}} \cup \delta^{\mathcal{B}}$.
- $F := F^{\mathcal{A}} \cup F^{\mathcal{B}}$.

It is easy to see that $\mathcal{L}(\mathcal{T}) = \mathcal{L}(\mathcal{A}) \cup \mathcal{L}(\mathcal{B})$ and that $\|\mathcal{T}\| = \|\mathcal{A}\| + \|\mathcal{B}\|$. The construction can also be easily implemented in time $O(|\Sigma| \times (\|\mathcal{A}\| + \|\mathcal{B}\|))$.

We now describe the construction for (2), which is also known as *product construction*. Define the NWA $\mathcal{T} = (\Sigma, Q, \delta, Q_0, F)$ as follows:

- $Q := Q^{\mathcal{A}} \times Q^{\mathcal{B}}$.
- $Q_0 := Q_0^{\mathcal{A}} \times Q_0^{\mathcal{B}}$.
- $\delta((q, q'), a) := \delta^{\mathcal{A}}(q, a) \times \delta^{\mathcal{B}}(q', a)$ for all $q \in Q^{\mathcal{A}}$, $q' \in Q^{\mathcal{B}}$, and $a \in \Sigma$.
- $F := F^{\mathcal{A}} \times F^{\mathcal{B}}$.

Intuitively, the automaton \mathcal{T} simulates both \mathcal{A} and \mathcal{B} on the given input word *simultaneously*. It is not hard to see that $\mathcal{L}(\mathcal{T}) = \mathcal{L}(\mathcal{A}) \cap \mathcal{L}(\mathcal{B})$ and $\|\mathcal{A}\| = \|\mathcal{A}\| \times \|\mathcal{B}\|$. Furthermore, this construction can be easily implemented in time $O(|\Sigma| \times \|\mathcal{A}\| \times \|\mathcal{B}\|)$. \square

We now state the following basic result on checking language emptiness and membership for NWAs. The proof of the following proposition can be done by using a simple reachability algorithm (e.g. see [Var95]).

Proposition 2.2.4 *Checking whether the language recognized by a given NWA \mathcal{A} is empty can be done in time $O(\|\mathcal{A}\| \times |\Sigma|)$. Consequently, checking whether a word $w \in \Sigma^*$ is a member of $\mathcal{L}(\mathcal{A})$ is solvable in time $O(|w| \times \|\mathcal{A}\| \times |\Sigma|)$.*

Star-free regular languages

Star-free regular languages form an important subclass of the class of regular languages. They are precisely the languages over the alphabet Σ generated *star-free regular expressions* over Σ , which are defined by the following grammars:

$$e, e' ::= \varepsilon \mid a \ (a \in \Sigma) \mid e + e' \mid e.e' \mid \bar{e}.$$

The semantics for $e + e'$ and $e.e'$ are the same as the regular expressions. We define $\mathcal{L}(\bar{e}) = \Sigma^* \setminus \mathcal{L}(e)$. Therefore, star-free regular expressions do not allow Kleene star operator, but instead allow complementation. Although standard regular expressions do not have built-in complementation operator, it is easy to see that they are definable using regular expressions (e.g. using Proposition 2.2.2). On the other hand, it is well-known that star-free regular languages actually form a proper subclass of the class of regular languages [MP71]. Other proofs of this result can also be found in [Lib04, Tho96]. We shall see later that there is a tight connection between star-free regular languages and the class of languages definable in first-order logic over finite words.

We now touch the computational complexity aspect of star-free regular expressions. The most important such result for the present thesis is that checking whether two star-free regular expressions over an alphabet consisting at least two letters generate the same language is decidable but is nonelementary, i.e., cannot be decided in k -fold exponential time for some integer $k > 0$.

Proposition 2.2.5 ([Sto74]) *The language equivalence problem for star-free regular expressions over an alphabet Σ with $|\Sigma| \geq 2$ is decidable but is nonelementary.*

This proposition has been commonly used in the literature for deriving fundamental complexity lower bounds for translations between automata and logic (cf. [Sto74, Tho96]).

2.2.2 Automata over ω -words

Languages over ω -words

Fix a finite alphabet Σ . An ω -word over Σ is a mapping w from $\mathbb{Z}_{>0}$ to Σ . As for finite words, we will often think of w as the infinite sequence $w(1)w(2)\dots$. Let Σ^ω denote the set of all ω -words over Σ . We use the notation $w[i, j]$ and for nonnegative integers $i \leq j$ to denote the finite word $w(i)\dots w(j)$. Similarly, $w[i, \infty)$ denotes the ω -word $w(i)w(i+1)\dots$. Given a finite word $w = a_1 \dots a_n \in \Sigma^*$ and an ω -word $w' \in \Sigma^\omega$, we define their concatenation as the ω -word $w.w'$ (also written as ww') as follows

$$(w.w')(i) := \begin{cases} a_i & \text{if } 1 \leq i \leq n \\ w'(i-n) & \text{if } i > n. \end{cases}$$

An ω -word language over the alphabet Σ is simply a subset of Σ^ω . As for finite words, we could apply the standard set operations (union, intersection, and complement) to ω -word languages. Given a finite word language $\mathcal{L} \subseteq \Sigma^*$ and an ω -word language $\mathcal{L}' \subseteq \Sigma^\omega$, we could define their *concatenation* as the ω -word language

$$\mathcal{L}.\mathcal{L}' := \{uv : u \in \mathcal{L}, v \in \mathcal{L}'\}.$$

We shall mostly write $\mathcal{L}\mathcal{L}'$ instead of $\mathcal{L}.\mathcal{L}'$ when the meaning is clear.

ω -regular languages

As in the case of finite words, we can define the notion of ω -regular languages as those ω -word languages that can be finitely represented by finite automata in some way. Let us now make this notion more precise. Given an NWA $\mathcal{A} = (\Sigma, Q, \delta, Q_0, F)$ and an ω -word $w \in \Sigma^\omega$, a *run of \mathcal{A} on w* is a function $\pi : \mathbb{N} \rightarrow Q$ such that $\pi(0) \in Q_0$ and, for each $i \in \mathbb{N}$, we have $(\pi(i), w(i+1), \pi(i+1)) \in \delta$. We say that π is said to be *accepting* if there exists infinitely many indices $i \in \mathbb{N}$ such that $\pi(i) \in F$. In other words, F is visited infinitely often in π . Such an acceptance condition is commonly referred to as *Büchi acceptance condition*. The ω -word w is *accepted* by \mathcal{A} if there exists an accepting run of \mathcal{A} on w . The language $\mathcal{L}(\mathcal{A})$ *accepted* by \mathcal{A} is simply the set of all ω -words $w \in \Sigma^\omega$ that are accepted by \mathcal{A} . When using NWAs as acceptors of ω -word languages,

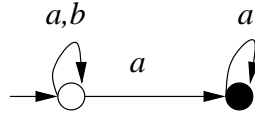


Figure 2.2: An NBWA recognizing the language $\{a, b\}^* \{a\}^\omega$.

we refer to such NWAs as *nondeterministic Büchi word-automata (NBWA)*. When the NWA is deterministic, we say that it is *deterministic Büchi word-automaton (DBWA)*. A language $\mathcal{L} \subseteq \Sigma^\omega$ is said to be ω -regular (or simply *regular*, when the context is clear) if it is accepted by some NBWA over Σ . For example, the language $\{a, b\}^* \{a\}^\omega$ is accepted by the NBWA depicted in Figure 2.2.

As in the case of finite words, ω -regular languages also satisfy desirable closure properties including union, intersection, and complementation. On the other hand, DBWAs are not as powerful as NBWAs, unlike in the case of finite words. For example, the language $\{a, b\}^* \{a\}^\omega$ accepted by the NBWA in Figure 2.2 cannot be accepted by a DBWA (for a proof, see [Var95]). Although there are deterministic automata models on ω -words that capture ω -regular languages, we will not encounter them in the sequel.

We now state a basic result on emptiness checking for languages recognized by NBWAs. The proof of the following proposition can be found in [Var95].

Proposition 2.2.6 *Checking whether a given NBWA \mathcal{A} recognizes an empty language can be done in linear time.*

Loosely speaking, the proof of the aforementioned proposition goes as follows. We first run Kosaraju’s linear-time algorithm (see [AHU83]) to find the strongly connected components of the NBWA \mathcal{A} (viewed as a directed graph). Checking emptiness, then, amounts to finding a path from some strongly connected component that contains an initial state to a strongly connected component that contains some final state q_F and *at least* one edge (so that there is a non-empty cycle that visits the final state q_F).

2.2.3 Automata over trees

A *direction alphabet* Υ is a nonempty downward-closed subset of the set $\mathbb{Z}_{\geq 1}$ of positive integers, i.e., if $i \in \Upsilon$ and $1 \leq j < i$, then $j \in \Upsilon$. Given a *direction alphabet* Υ , a *tree domain* over Υ is a non-empty set $D \subseteq \Upsilon^*$ that satisfies the following two properties:

- D is prefix-closed, i.e., for each $w \in \Upsilon^*$ and $i \in \Upsilon$, $wi \in D$ implies $w \in D$, and

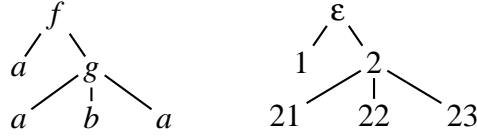


Figure 2.3: An example of a 3-ary tree over $\{f, g, a, b\}$ (left) depicted together with its tree domain (right) whose elements are ordered by the prefix-of relations.

- for all $w_i \in D$ and $1 \leq j < i$, it is the case that $w_j \in D$.

For a nonempty set Σ called a *labeling alphabet*, a *tree* over Σ with direction alphabet Υ is a pair $T = (D, \tau)$ where D is a tree domain over Υ and τ is a function mapping D to Σ . A k -ary tree over Σ is a tree over Σ with direction alphabet $\Upsilon = \{1, \dots, k\}$. Unless otherwise stated, we shall say “trees” to mean k -ary trees, for some positive integer k , over a *finite* labeling alphabet with a *finite* tree domain. Figure 2.3 gives an example of a tree depicted together with its tree domain. When the meaning is clear, we shall omit mention of Υ and simply say that T is a tree over Σ (or just “tree” when Σ is clear).

We shall now define some standard graph-theoretic terminologies for dealing with trees. Fix a k -ary tree $T = (D, \tau)$ over the labeling alphabet Σ . The elements of D are also called *nodes*. Therefore, we shall call τ a *node labeling* and that each node $u \in D$ is *labeled* by $\tau(u)$. The *level* of a node $u \in D$, denoted by $\text{level}(u)$, is simply the length $|u|$ of the word u . The *height* of the tree T is defined as $1 + \max\{\text{level}(u) : u \in D\}$. We shall refer to the *root* of T as the node $\varepsilon \in D$. Words $u \in D$ such that no ui is in D are called *leaves*. If $v \in D$ and $vi \in D$ for some $i \in \Upsilon$, then we call vi a *child* of v and v the *parent* of vi . Likewise, if $vi \in D$ and $vj \in D$ for some $i, j \in \Upsilon$, then we say that vi and vj are *siblings*. In addition, if $v, vw \in D$ for some $w \in \Upsilon^*$, then vw is a *descendant* of v and v an *ancestor* of vw . A *path* (or *branch*) starting at a node $v \in D$ is simply a sequence $\pi = v_0, \dots, v_n$ of nodes such that $v_0 = v$ and v_i is a child of v_{i-1} for each $i = 1, \dots, n$. The tree T is said to be *complete* if, whenever $u \in D$ and $ui \in D$ for some $i \in \Upsilon$, it is the case that $uj \in D$ for all $j \in \Upsilon$.

The notion of “prefix” for words has a natural analogue for trees. We shall define this next. For k -ary trees $T = (D, \tau)$ and $T' = (D', \tau')$ over the labeling alphabet Σ , we say that T *extends* T' , written $T' \preceq T$, iff $D' \subseteq D$ and, for each $u \in D'$, we have $\tau(u) = \tau'(u)$. Observe that the relation \preceq reduces to the prefix-of relations in the word case. In the sequel, we call the relation \preceq on $\text{TREE}_k(\Sigma)$ to be the *tree extension relation*.

We now define the notion of “subtrees”, which is the tree analogue of the standard notion of “suffix” of a word. Given k -ary trees $T_1 = (D_1, \tau_1)$ and $T_2 = (D_2, \tau_2)$ over the

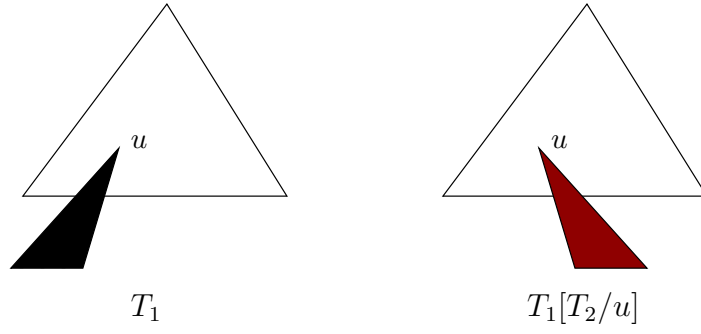


Figure 2.4: A depiction of the subtree substitution operation. Here the subtree rooted at u on the left is replaced by the tree T_2 , which is the subtree rooted at u on the right.

labeling alphabet Σ , we say that T_2 is a *subtree of T_1 rooted at $u \in D_1$* if D_2 coincides with $\{v : uv \in D\}$ and that $\tau_2(v) = \tau_1(uv)$ for each $v \in D_2$. This also motivates the *substitution operation* on subtrees. Given k -ary trees $T_1 = (D_1, \tau_1)$ and $T_2 = (D_2, \tau_2)$ over the labeling alphabet Σ and a node $u \in D_1$, we write $T_1[T_2/u]$ for the tree obtained by replacing the subtree of T_1 rooted at u by T_2 . More precisely, the tree $T_1[T_2/u]$ is defined to be the tree $T = (D, \tau)$ where

$$D := (D_1 \setminus \{uv \in D_1 : v \in \{0, \dots, 1\}^*\}) \cup uD_2,$$

$$\tau(w) := \begin{cases} \tau_1(w) & \text{if } w \in (D_1 \setminus \{uv \in D_1 : v \in \{0, \dots, 1\}^*\}), \\ \tau_2(v) & \text{if } w = uv \in uD_2. \end{cases}$$

See Figure 2.4 for an illustration. The subtree substitution operation can also be easily generalized to take into account multiple substitutions. Given k -ary trees T_0, \dots, T_n and nodes u_1, \dots, u_n in T_0 that are incomparable with respect to the prefix-of relations over words $\{1, \dots, k\}^*$ (i.e. no u_i is an ancestor of u_j for all distinct indices i, j), we define $T_0[T_1/u_1, \dots, T_n/u_n]$ to be the tree $(\dots(T_0[T_1/u_1])\dots)[T_n/u_n]$ obtained by applying the subtree substitution operations for multiple times (the order of which is of no importance).

The set of all k -ary trees over Σ is denoted by $\text{TREE}_k(\Sigma)$. A *tree language* over $\text{TREE}_k(\Sigma)$ is simply a subset of $\text{TREE}_k(\Sigma)$. Observe that word languages can be thought of as a subset of $\text{TREE}_1(\Sigma)$.

Regular tree languages

To define the notion of regular tree languages, we shall review a standard definition of tree automata. A (*top-down nondeterministic*) *tree automaton* over $\text{TREE}_k(\Sigma)$ is a tuple $\mathcal{A} = (\Sigma, Q, \delta, Q_0, F)$ where

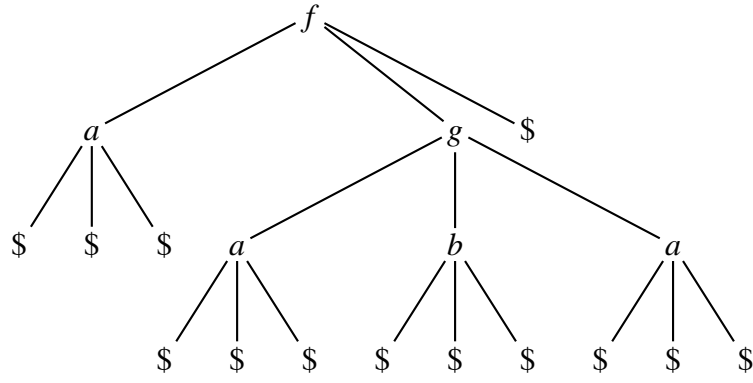


Figure 2.5: The tree depicted here is $\mathbf{virt}(T)$, where T is the tree from Figure 2.3.

- Q is a finite set of *states*,
- $Q_0 \subseteq Q$ is a set of *initial* states,
- $F \subseteq Q$ is a set of *final* states, and
- δ is a *transition relation*, i.e., a subset of $Q \times \Sigma \times Q^k$.

Note that the parameter k is *implicit* from the representation of \mathcal{A} , i.e., it can be deduced by inspecting the transition relation δ . In the sequel, we denote by $\mathbf{States}(\mathcal{A})$ the set of states of \mathcal{A} . For our constructions, it will be most convenient to define runs on trees by attaching “virtual” leaves. Given a k -ary tree $T = (D, \tau)$, we define $\mathbf{virt}(T)$ to be the k -ary tree (D', τ') over the alphabet $\Sigma' := \Sigma \cup \{\$, \}$, where $\$ \notin \Sigma$, such that $D' = D \cup \{vi : v \in D, 1 \leq i \leq k\}$ and

$$\tau'(u) = \begin{cases} \tau(u) & \text{if } u \in D, \\ \$ & \text{otherwise.} \end{cases}$$

See Figure 2.5 for an example. Notice that $\mathbf{virt}(T)$ is a complete tree. A *run* of \mathcal{A} on T then is a mapping $\rho : D' \rightarrow Q$ such that $\rho(\epsilon) \in Q_0$ and, for each node $u \in D'$ with children $u1, \dots, uk \in D'$, we have $(\rho(u1), \dots, \rho(uk)) \in \delta(\rho(u), \tau(u))$. A run is said to be *accepting* if $\rho(u) \in F$ for each leaf $u \in D'$. A tree $T \in \text{Tree}_k(\Sigma)$ is said to be *accepted* by \mathcal{A} if there exists an accepting run of \mathcal{A} on T . The language $\mathcal{L}(\mathcal{A})$ recognized by \mathcal{A} is the set of k -ary trees over Σ accepted by \mathcal{A} . Such a language is said to be *tree-regular*. In the sequel, we shall abbreviate nondeterministic tree automata as NTA.

For the purpose of complexity analysis, we shall now define the *size* $\|\mathcal{A}\|$ of an NTA $\mathcal{A} = (\Sigma, Q, \delta, q_0, F)$ over k -ary trees. Let $\|\mathcal{A}\|$ be the number of tuples $(q, q_1, \dots, q_k) \in$

Q^{k+1} such that $(q, a, q_1, \dots, q_k) \in \delta$ for some $a \in \Sigma$. As for automata over finite words, without loss of generality, we may assume that each state in Q occurs in δ at least once. In this case, the number of bits needed to represent \mathcal{A} is at most

$$O(\|\mathcal{A}\| \times |\Sigma| \times k \log(|Q|) \times \log(|\Sigma|)),$$

which is polynomial in both $\|\mathcal{A}\|$ and $|\Sigma|$. This justifies our definition of $\|\mathcal{A}\|$.

Remark 2.2.2 Several important remarks are in order. Firstly, there is also a notion of *bottom-up nondeterministic tree automata*, which recognize precisely regular tree languages (cf. [CDG⁺07]). Furthermore, the translations between these two representations can be performed in linear time. Secondly, it is useful to keep in mind that, although we may define the *deterministic* tree automata with respect to these two flavors of tree automata, only the bottom-up notion gives the full power of regular tree languages. Since we will not need it in the sequel, we shall avoid further mention of deterministic tree automata. Finally, we cannot simply assume that we only deal with NTAs with only one final state (unlike in the case of word automata). More precisely, the conversion from general NTAs to those with only one final state might cause an exponential blow-up in the size of the direction alphabet. Incidentally, if we have “ ε -transitions” in our NTAs (which are abundant, among others, in the literature of ground tree rewrite systems and ground tree transducers [CDG⁺07, Löd03, Löd06]), the polynomial-time procedure of removing these ε -transitions naturally yield NTAs with multiple final states (see below). In contrast, it is possible to assume that an NTA has only one *initial* state. This is because we can introduce a new initial state q_0 and add *linearly* many extra transitions in the standard way. In the sequel, we shall often assume that NTAs have only one initial state and write $(\Sigma, Q, \delta, q_0, F)$ instead of $(\Sigma, Q, \delta, \{q_0\}, F)$. ■

Some basic results

Regular tree languages satisfies the same closure properties that are satisfied by regular word languages, e.g., union, intersection, and complementation. The following proposition can be proved in the same way as Proposition 2.2.3 (see [CDG⁺07] for a proof).

Proposition 2.2.7 *Given NTAs \mathcal{A} and \mathcal{B} over $\text{TREE}_k(\Sigma)$:*

- *we can compute in time $O(|\Sigma| \times (\|\mathcal{A}\| + \|\mathcal{B}\|))$ an NTA of size $\|\mathcal{A}\| + \|\mathcal{B}\|$ recognizing the language $\mathcal{L}(\mathcal{A}) \cup \mathcal{L}(\mathcal{B})$,*

- we can compute in time $O(|\Sigma| \times \|\mathcal{A}\| \times \|\mathcal{B}\|)$ an NTA of size $\|\mathcal{A}\| \times \|\mathcal{B}\|$ recognizing the language $\mathcal{L}(\mathcal{A}) \cap \mathcal{L}(\mathcal{B})$, and
- if $|\text{States}(\mathcal{A})| = n$, we can compute in exponential time an NTA with 2^n states recognizing the language $\overline{\mathcal{L}(\mathcal{A})}$.

Checking language emptiness (and hence membership) for tree automata is also easy, as in the case of word languages. The proof of the following proposition can be found in [CDG⁺07].

Proposition 2.2.8 *Checking whether an NTA $\mathcal{A} = (\Sigma, Q, \delta, Q_0, F)$ over k -ary trees (where k is not fixed) recognizes a non-empty language can be done in time $O(\|\mathcal{A}\| \times |\Sigma|)$. Consequently, checking whether a tree $T = (D, \tau) \in \text{TREE}_k(\Sigma)$ is a member of $\mathcal{L}(\mathcal{A})$ is solvable in time $O(|D| \times \|\mathcal{A}\| \times |\Sigma|)$.*

Nondeterministic tree automata with ε -transitions

We shall now introduce an extension of NTAs with ε -transitions. This is only done for the purpose of convenience when describing the NTAs. Roughly speaking, ε -transitions are transitions of the form (q, q') for a pair of states of the automaton. Intuitively, if we imagine a top-down tree automaton that runs on a tree T , then at any given node u of T when the automaton is at state q it can use the transition (q, q') to *instantaneously* switch to the state q' at the same node u . More precisely, an ε -NTA \mathcal{A} over $\text{TREE}_k(\Sigma)$ is a tuple $(\Sigma, Q, \delta, Q_0, F)$, where Σ , Q , Q_0 , and F are the same as for NTAs and

- δ is a *transition relation* containing transitions of the form $(q, a, q_1, \dots, q_k) \in Q \times \Sigma \times Q^k$, or of the form $(q, q') \in Q \times Q$.

Before defining the language $\mathcal{L}(\mathcal{A})$ accepted by the ε -NTA \mathcal{A} above, we let \Rightarrow denote the transitive closure of $\{(q, q') \in Q \times Q : (q, q') \in \delta\}$. Define a new NTA (without ε -transitions) $\mathcal{A}' := (\Sigma, Q, \delta', Q_0, F')$ as follows:

- $\delta' = \{(q, a, q_1, \dots, q_k) : \exists q' \in Q \text{ such that } (q', a, q_1, \dots, q_k) \in \delta \text{ and } q \Rightarrow q'\}$, and
- $F' := \{q : \exists q' \in F \text{ such that } q \Rightarrow q'\}$.

A tree T is said to be *accepted by* \mathcal{A} if it is accepted by \mathcal{A}' . The *language* $\mathcal{L}(\mathcal{A})$ of \mathcal{A} is defined to be the language $\mathcal{L}(\mathcal{A}')$ of \mathcal{A}' . The following proposition is now immediate.

Proposition 2.2.9 *Given an ε -NTA \mathcal{A} over $\text{TREE}_k(\Sigma)$ with n states, we may construct an NTA \mathcal{A}' over $\text{TREE}_k(\Sigma)$ of size $n \times \|\mathcal{A}\|$ such that $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}')$ in time $O(|\Sigma| \times n \times \|\mathcal{A}\|)$.*

Notice that the parameter k , which is *not fixed* for the problem, does not get into the exponent for the size and the computation time of the NTA \mathcal{A}' . In the sequel, we shall use ε -NTA solely for a descriptive purpose.

2.2.4 Automata over infinite trees

An *infinite k -ary tree* over the labeling alphabet Σ is a tuple $T = (D, \tau)$, where $D = \{1, \dots, k\}^*$ and τ is a mapping from D to Σ . Let $\text{TREE}_k^\omega(\Sigma)$ denote the class of all infinite k -ary trees over Σ .

Büchi-recognizable infinite-tree languages

In the sequel, we do not need the full power of regular infinite tree languages, which are usually defined by automata over infinite trees with powerful acceptance conditions such as Rabin, parity, and Muller [Tho96]. We shall only need automata over infinite trees with Büchi accepting condition, which is well-known to be strictly less powerful than regular infinite-tree languages (e.g. see [Tho96]) unlike in the case of ω -word automata.

A (*nondeterministic*) *Büchi k -ary tree automaton*, abbreviated as NBTA, over Σ is a k -ary tree automaton $\mathcal{A} = (\Sigma, Q, \delta, q_0, F)$ that we defined for finite trees, except with infinite trees as input. Given an infinite-tree $T = (\{1, \dots, k\}^*, \tau)$, a *run* of \mathcal{A} on T is a mapping $\rho : \{1, \dots, k\}^* \rightarrow Q$ such that $\rho(\varepsilon) = q_0$ and, for each node $v \in \{1, \dots, k\}^*$, we have $(\rho(v1), \dots, \rho(vk)) \in \delta(\rho(v), \tau(v))$. The run ρ is said to be accepting if, for each infinite path $\pi = \{v_i\}_{i=0}^\infty$ in the run ρ (viewed as an infinite k -ary tree) starting at the root ε , there exists infinitely many indices i such that $\rho(v_i) \in F$. In other words, for each infinite path in ρ starting at the root, the Büchi acceptance condition for ω -words is satisfied. The language $\mathcal{L}(\mathcal{A})$ recognized by \mathcal{A} is the set of all k -ary infinite-trees over Σ that are accepted by \mathcal{A} . Such an infinite-tree language is said to be *Büchi-recognizable*. The proof of the following proposition can be found in [VW86b], although it was first proved in [Rab70] for binary trees.

Proposition 2.2.10 *Checking whether a given Büchi tree automaton recognizes a non-empty language can be done in quadratic time.*

2.2.5 Pushdown automata and context-free grammars

We now turn back to languages over finite words. A *context-free grammar* (CFG) \mathcal{G} over the alphabet Σ is a 4-tuple (Σ, V, δ, S_0) where

- V is a set of *non-terminals*,
- $S_0 \in V$ is an initial non-terminal,
- δ is the set of *rewrite rules*, which is a finite subset of $V \times (V \times \Sigma)^*$

When discussing context-free grammars, the elements of Σ are also often called *terminals*. Given two sequences $\alpha, \beta \in (V \times \Sigma)^*$ of non-terminals and terminals, we say that β can be *immediately derived* from α , written $\alpha \Rightarrow \beta$, if there exist words $u, v \in (V \times \Sigma)^*$ and a rewrite rule $(X, w) \in \delta$ such that $\alpha = uXv$ and $\beta = uwv$. Let \Rightarrow^* denote the transitive-reflexive closure of this immediate derivation relation $\Rightarrow \subseteq (V \times \Sigma)^* \times (V \times \Sigma)^*$. We say that α can be *derived* from β if $\alpha \Rightarrow^* \beta$. A sequence $v \in \Sigma^*$ of terminals is said to be *derivable* by \mathcal{G} if $S_0 \Rightarrow^* v$. The language $\mathcal{L}(\mathcal{G})$ generated by \mathcal{G} is simply the set of words $v \in \Sigma^*$ that are derivable by \mathcal{G} . A language $\mathcal{L} \subseteq \Sigma^*$ is said to be *context-free* if some CFG \mathcal{G} generates \mathcal{L} . It is well-known that context-free languages strictly subsume regular languages.

We now define pushdown automata, which are another model with the same expressive power as context-free grammars. Fix an (input) alphabet Σ and let $\Sigma_\epsilon := \Sigma \cup \{\epsilon\}$. A *pushdown automaton* (PDA) \mathcal{P} over the input alphabet Σ is a tuple $(\Sigma, \Gamma, Q, \delta, q_0, F)$ where:

- Γ is a finite *stack alphabet* containing the special *stack-bottom symbol* $\$ \in \Gamma$,
- Q is a finite set of *states*,
- q_0 is an *initial state*,
- $F \subseteq Q$ is a set of *final states*, and
- δ is a *transition relation*, which is a finite subset of $(Q \times \Sigma_\epsilon \times \Gamma \cup \{\epsilon\}) \times (Q \times \Gamma^*)$.

The PDA \mathcal{P} is said to be ϵ -free if δ is a subset of $(Q \times \Sigma \times \Gamma) \times (Q \times \Gamma^*)$. A *stack content* (with respect to \mathcal{P}) is a word $w \in \Sigma^*$. The topmost symbol of the stack is on the right¹. A *configuration* of \mathcal{P} is a pair (q, w) of state $q \in Q$ and a *stack content*

¹In the literature, stack contents are often written in the reversed way, i.e., topmost symbol on the left. As we shall see later, this convention is more suitable for our purposes.

w . Given two configurations (q, w) and (q', w') of \mathcal{P} and the symbol $a \in \Sigma_\epsilon$, we write $(q, w) \rightarrow_a (q', w')$ if there exists a word $v \in \Gamma^*$ such that, for some words $u \in \Gamma \cup \{\epsilon\}$ and $u' \in \Gamma^*$ we have $w = vu$ and $w' = vu'$ and that $((q, a, u), (q', u'))$ is a transition in \mathcal{P} . Given an input word $v \in \Sigma^*$, we write $(q, w) \rightarrow_v (q', w')$ if there exists a sequence $a_1, \dots, a_n \in \Sigma_\epsilon$ of input letters (possibly interleaved with empty words) such that $v = a_1 \dots a_n$ and there exists a sequence of configurations $(q_0, w_0), \dots, (q_n, w_n)$ of \mathcal{P} such that $(q_0, w_0) = (q, w)$, $(q_n, w_n) = (q', w')$, and

$$(q_0, w_0) \rightarrow_{a_1} \dots \rightarrow_{a_n} (q_n, w_n).$$

We say that \mathcal{P} *accepts* the word $v \in \Sigma^*$ if, for some final state $q_F \in F$, we have $(q_0, \$) \rightarrow_v (q_F, \$)$. The *language* $\mathcal{L}(\mathcal{P})$ of \mathcal{P} is the set of words $v \in \Sigma^*$ that are accepted by \mathcal{P} . We say also that \mathcal{P} *accepts* $\mathcal{L}(\mathcal{P})$. The following proposition is well-known (e.g. see [Sip97] for a proof).

Proposition 2.2.11 *There exists a polynomial-time algorithm, which given a CFG \mathcal{G} over Σ , computes a PDA \mathcal{P} over Σ such that $\mathcal{L}(\mathcal{G}) = \mathcal{L}(\mathcal{P})$. Conversely, there exists a polynomial-time algorithm, which given a PDA \mathcal{P} over Σ , computes a CFG \mathcal{G} over Σ such that $\mathcal{L}(\mathcal{G}) = \mathcal{L}(\mathcal{P})$.*

2.2.6 Parikh's Theorem and semilinear sets

We shall now recall Parikh's Theorem [Par66] — one of the most celebrated theorem in automata theory — and its connection to *semilinear sets*. Roughly speaking, Parikh's Theorem states that the sets of letter-counts (a.k.a. Parikh images) of regular languages and context-free languages precisely coincide with semilinear sets. Let us now make these more precise.

Let us tacitly assume that our finite input alphabet $\Sigma = \{a_1, \dots, a_k\}$ has some total ordering \prec , say $a_1 \prec \dots \prec a_k$. Given a word $w \in \Sigma$, we let $\mathcal{P}(w)$ be the tuple $(|w|_{a_1}, \dots, |w|_{a_k}) \in \mathbb{N}^k$. In other words, $\mathcal{P}(w)$ is obtained by “forgetting” the ordering of the word w , i.e., only count multiplicities of each letter in the alphabet Σ . The *Parikh image* $\mathcal{P}(\mathcal{L})$ of the language $\mathcal{L} \subseteq \Sigma^*$ is simply the set $\{\mathcal{P}(w) : w \in \mathcal{L}\} \subseteq \mathbb{N}^k$. This definition allows us to now talk about the Parikh images of (the languages of) CFGs and NWAs.

Let us now recall the definition of *semilinear sets*. For every vector $\mathbf{v} \in \mathbb{Z}^k$ and every finite set $S = \{\mathbf{u}_1, \dots, \mathbf{u}_m\}$ of vectors in \mathbb{Z}^k , we write $P(\mathbf{v}; S)$ to denote the *\mathbb{Z} -linear set* $\{\mathbf{v} + \sum_{i=1}^m a_i \mathbf{u}_i : a_1, \dots, a_m \in \mathbb{N}\}$. The pair $B := \langle \mathbf{v}; S \rangle$ is said to be a *linear*

basis for $P(\mathbf{v}; S)$. Notice that there exist non-unique linear bases for a \mathbb{Z} -linear set. The vector \mathbf{v} is said to be the *offset* of B , and the vectors S the *periods* (or *generators*) of B . A \mathbb{Z} -semilinear set S is simply a finite (possibly empty) union of \mathbb{Z} -linear sets $P(\mathbf{v}_1; S_1), \dots, P(\mathbf{v}_s; S_s)$. In this case, we say that $\mathcal{B} = \{\langle \mathbf{v}_i; S_i \rangle\}_{i=1}^s$ is a *semilinear basis* for $P(\mathcal{B}) := S$. Likewise, semilinear bases for S are not unique. A \mathbb{Z} -semilinear set $S \subseteq \mathbb{Z}^k$ is said to be \mathbb{N} -semilinear (or simply *semilinear*) if it has a semilinear basis with vectors from \mathbb{N}^k only. The notion of \mathbb{N} -linear (or simply *linear*) sets is also defined similarly. In the sequel, we shall *not* distinguish (semi)linear sets and their bases, when it is clear from the context. Thus, we shall use such a phrase as “compute a (semi)linear set” to mean that we compute a particular (semi)linear basis for it.

Remark 2.2.3 Since \mathbb{Z} -(semi)linear bases \mathcal{B} are simply a sequence of vectors from \mathbb{Z}^k , we could talk about their size $\|\mathcal{B}\|$ when represented on the tapes of Turing machines. In particular, we shall use both *unary* and *binary* representations of numbers, and be explicit about this when necessary. ■

The connection between Parikh images of CFGs and NWA and semilinear sets is given by Parikh’s Theorem, which we shall state next.

Theorem 2.2.12 (Parikh [Par66]) *Given a subset $S \subseteq \mathbb{N}^k$, the following statements are equivalent:*

- (1) S is a semilinear set.
- (2) $S = \mathcal{P}(L(\mathcal{A}))$ for some NWA \mathcal{A} over $\Sigma = \{a_1, \dots, a_k\}$.
- (3) $S = \mathcal{P}(L(G))$ for some CFG G over $\Sigma = \{a_1, \dots, a_k\}$.

Furthermore, the translations among these finite representations of the set S are effective.

The translations from (1) to (2) is rather obvious, which can be done in polynomial time (provided that we represent numbers in unary). There exists a simple polynomial-time algorithm which, given an NWA, computes an equivalent CFG (e.g. see [Sip97]) yielding a translation from (2) to (3). The non-trivial part is the translation from (3) to (1), which was given by Parikh [Par66]. We shall mention also that there are other constructions from (3) to (1) with different flavors and techniques (cf. [Esp97b, Koz97, SSMH04, VSS05]). All these constructions run in exponential time (in fact, they produce exponentially many linear sets in the worst case). We shall see in Chapter 7

that this cannot be improved even for CFGs over the fixed alphabet $\Sigma = \{a\}$. On the other hand, we shall give a direct translation from (2) to (1) in Chapter 7 that has a polynomial-time worst case complexity when the size of the alphabet is fixed.

2.3 Computability and complexity theory

In this section, we shall recall briefly some standard concepts from computability and complexity theory (e.g. see [Koz97, Koz06, Pap94, Sip97] for more details).

2.3.1 Computability

For the sake of completeness, we shall briefly recall the definition of Turing machines. A *nondeterministic Turing machine* is a tuple $\mathcal{M} = (\Sigma, \Gamma, Q, \delta, q_0, q_F, \square)$, where the following are satisfied:

- Σ is an *input alphabet*.
- Γ is a *stack alphabet* satisfying $\Sigma \cup \{\square\} \subseteq \Gamma$. Here \square is a reserved *blank symbol*.
- Q is a set of *states*.
- $q_0 \in Q$ is an *initial state*.
- $q_F \in Q$ is an *accept state*.
- $\delta: (Q \times \Gamma) \times (Q \times \Gamma \times \{L, R\})$ a *transition function*.

The machine \mathcal{M} is said to be *deterministic* if δ is a function from $Q \times \Gamma$ to $Q \times \Gamma \times \{L, R\}$. In the sequel, a Turing machine is deterministic, unless stated otherwise. The configuration of \mathcal{M} is simply a word in the language $\Gamma^*(Q \times \Gamma)\Gamma^*$. The one-step reachability relation \rightarrow between configurations of \mathcal{M} can be defined in the standard way: $((q, a), (q', b, d)) \in \delta$ is executed iff, whenever the machine is in state q and the tape cell currently pointed to by the machine holds the value a , then the machine rewrites the value a by b , switches to state q' , and moves the pointer to the left if $d = L$ and to the right if $d = R$. Without loss of generality, we assume that there is no transition from q_F . The machine is said to be *halting* on a word w if every run of \mathcal{M} starting from the configuration $(q, \square)w$ eventually reaches the state q_F or a dead end. The machine is said to be *halting* if it is halting on every input word. For the case when there exists a run of \mathcal{M} that reaches q_F on an input word w , we say that \mathcal{M} *accepts* the

word w . Otherwise, it rejects w . The language $\mathcal{L}(\mathcal{M})$ *accepted* by \mathcal{M} consists of all words $w \in \Sigma^*$ which are accepted by \mathcal{M} .

A language \mathcal{L} is said to be *recursively enumerable (r.e.)* if it is accepted by a Turing machine. It is said to be *co-recursive-enumerable (resp. co-r.e.)* if its complement is accepted by a Turing machine. It is said to be *recursive (or decidable)* if it is accepted by a halting Turing machine \mathcal{M} . In this case, we also say that \mathcal{L} is *decided* by \mathcal{M} . Let Σ_1^0 (resp. Π_1^0) denote the class of r.e. (co-r.e.) languages. Let Δ_1^0 denote the class of decidable languages. For any \mathcal{C} of these sets, a language \mathcal{L} is said to be \mathcal{C} -hard if for each language $\mathcal{L}' \in \mathcal{C}$ there exists a halting Turing machine \mathcal{M} which given an input word w outputs another word $\mathcal{M}(w)$ such that $w \in \mathcal{L}'$ iff $\mathcal{M}(w) \in \mathcal{L}$. Such a Turing machine is also said to be a *(many-one) reduction*. The language \mathcal{L} is said to be \mathcal{C} -complete if it is in \mathcal{C} and is \mathcal{C} -hard.

It is well-known that $\Sigma_1^0 \cap \Pi_1^0 = \Delta_1^0$, but $\Sigma_1^0 \neq \Pi_1^0$. These sets can be generalized in a natural way to form an *arithmetic hierarchy* (cf. [Koz06]). There are languages that are beyond this hierarchy. In the sequel, we shall briefly see languages that are Σ_1^1 -complete. In recursion theory, Σ_1^1 can be understood as the class of all relations $R \subseteq \mathbb{N}^n$ that can be defined by a formula of the form

$$\exists f \varphi(x_1, \dots, x_n, f),$$

where f ranges over number-theoretic functions (i.e. domains and co-domains are \mathbb{N}) and φ is a first-order formula in number theory (i.e. quantification over numbers). Σ_1^1 -complete languages are also often said to be *highly undecidable*.

Remark 2.3.1 As often done in computability theory, the term “languages” will henceforth be synonymously identified with “problems”.

2.3.2 Complexity theory

In computational complexity, we restrict ourselves to decidable languages and try to classify difficulty of these languages by restricting resources such as time, space, and nondeterminism. Given a halting Turing machine \mathcal{M} and an input word w , we can measure the time (or the number of steps) or space that are required to reach a halt. The time (resp. space) complexity of a Turing machine is then defined to be the function $f : \mathbb{N} \rightarrow \mathbb{N}$ such that for every $n \in \mathbb{N}$ the maximum amount of time (resp. space) required by \mathcal{M} before it terminates on an input word of length n is $f(n)$.

Let us now recall some standard complexity classes. For a function $f : \mathbb{N} \rightarrow \mathbb{N}$, we define $\text{DTIME}(f(n))$ (resp. $\text{NTIME}(f(n))$) to be the class of problems solvable by a deterministic (resp. nondeterministic) Turing machine that runs in time $O(f(n))$. Similarly, we define $\text{SPACE}(f(n))$ to be the class of problems that are solvable by a Turing machine that uses $O(f(n))$ space. Let us define the class exp_k of functions for every $k \in \mathbb{N}$ by induction. Let exp_0 be the class of all polynomial functions. By induction, the class exp_k ($k > 0$) contains all functions of the form $2^{O(f(n))}$, where f is a function in exp_{k-1} . By convention, exp stands for exp_1 . We may now define the following standard complexity classes:

- P is the class of problems solvable in polynomial-time.
- PSPACE is the class of problems solvable in polynomial space.
- NP is the class of problems solvable by nondeterministic polynomial-time Turing machines.
- NPSPACE is the class of problems solvable by nonterministic polynomial-space Turing machines.
- $k\text{-EXP}$ is the class of problems that are solvable in time exp_k .
- EXP is the class 1-EXP .

The following containments are standard:

$$P \subseteq \text{NP} \subseteq \text{PSPACE} = \text{NSPACE} \subseteq \text{EXP} \subset 2\text{-EXP} \subset \dots$$

Here, set containments of the form \subseteq are not known to be strict. Each of these complexity classes have complete problems under many-one reductions that run in polynomial time. A decidable problem is said to be *elementary* if it is in $k\text{-EXP}$ for some $k \in \mathbb{N}$. Otherwise, it is said to be *nonelementary*.

For a complexity class C , we write $\text{co}C$ for the set of problems whose complements are solvable in C . For example, coNP is the set of problems whose complements are in NP . While the classes P , PSPACE , and $k\text{-EXP}$ closed under complements, it is not known whether $\text{NP} = \text{coNP}$.

We now define polynomial hierarchy. Let

$$\Delta_0^P = \Sigma_0^P = \Pi_0^P = P.$$

For each $k > 0$, let

$$\begin{aligned}\Delta_k^P &= P^{\Sigma_{k-1}^P}, \\ \Sigma_k^P &= NP^{\Sigma_{k-1}^P}, \\ \Pi_k^P &= coNP^{\Sigma_{k-1}^P}.\end{aligned}$$

Here, the notation $P^{\Sigma_{k-1}^P}$ refers to the class of problems solvable by a polynomial-time machine that can make calls to an oracle solving a problem in Σ_{k-1}^P . Oracles are viewed as blackboxes and so their computation time is measured as a constant. The notations $NP^{\Sigma_{k-1}^P}$ and $coNP^{\Sigma_{k-1}^P}$ can also be defined in a similar way. Each of these classes are known to have complete problems. Let PH be the union of all Σ_k^P ($k \in \mathbb{N}$). It is known that $PH \subseteq PSPACE$. In Chapter 8, we will see the complexity classes P^{NP} and $P^{NP[\log]}$. The former is simply the class Δ_2^P , while the latter is the class of problems solvable by polynomial-time Turing machines that can only make logarithmically many calls to NP oracles. Clearly, we have $P^{NP[\log]} \subseteq P^{NP}$. The class $P^{NP[\log]}$ is also known to contain NP, coNP, and even the entire *boolean hierarchy*. We shall only mention the first level of boolean hierarchy, which is the class DP containing problems of the form

$$\{\langle v, w \rangle : v \in \mathcal{L}, w \in \mathcal{L}'\}.$$

for some NP language \mathcal{L} and coNP language \mathcal{L}' .

We shall now briefly recall the notion of alternating Turing machines (see [Koz06, Sip97] for more details). Alternating Turing machines can be viewed as generalizations of nondeterministic Turing machines with universal states. More precisely, each state of a Turing machine \mathcal{M} is declared either existential or universal. To accept from a configuration $\mathbf{c} = v(q, a)w$ of \mathcal{M} where q is existential, some runs of \mathcal{M} from \mathbf{c} will have to result in an accept. On the other hand, to accept from a configuration $\mathbf{c} = v(q, a)w$ of \mathcal{M} where q is universal, *all* runs of \mathcal{M} from \mathbf{c} will have to result in an accept. As before, we can define the amount of time/space used by the algorithm on an input word w and define the time/space used by the algorithm in terms of its worst-case complexity. We write $ATIME(f(n))$ (resp. $ASPACE(f(n))$) to denote the class of problems solvable by an alternating Turing machine in time (resp. space) $O(f(n))$. In the sequel, we will meet the complexity classes:

- $AP = ATIME(\exp_0(n))$.
- $ALOG = ASPACE(\log(n))$.

- $\text{APSPACE} = \text{ASPACE}(\exp_0(n))$.

It is known that $\text{AP} = \text{PSPACE}$, $\text{ALOG} = \text{P}$, and $\text{APSPACE} = \text{EXP}$. Furthermore, poly-time alternating Turing machines with a fixed number of alternations (between existential states and universal states) in all their possible runs accept only languages in PH .

Finally, following [Koz06], we define the notation

$$\text{STA}(f_1(n), f_2(n), f_3(n))$$

to denote the class of problems solvable in space $f_1(n)$, in time $f_2(n)$ and with $f_3(n)$ alternations. We also write $*$ to denote unbounded. For example, we have the following:

- $\text{STA}(\exp_0(n), *, *) = \text{PSPACE}$,
- $\text{STA}(*, \exp_0(n), *) = \text{P}$, and
- $\text{STA}(*, \exp_0(n), \exp_0(n)) = \text{AP}$.

2.4 Structures and transition systems

In this section, we recall the standard definitions of (logical) structures and transition systems. For a more thorough treatment, the reader may consult the following references [BBF⁺01, BdRV01, CGP99, Lib04, Sti01, Tho96, vD08].

A *vocabulary* is a finite set $\sigma = \{a_1, \dots, a_n\}$ of (*relation*) *names* together with a function $\text{AR} : \sigma \rightarrow \mathbb{Z}_{>0}$ mapping each relation name to a positive integer representing its *arity*. A σ -*structure* \mathfrak{S} is a tuple $\langle S, \{R_a\}_{a \in \sigma} \rangle$ where S is some set (a.k.a. *universe* or *domain*) and $R_a \subseteq S^{\text{AR}(a_i)}$ is an $\text{AR}(a_i)$ -ary relation on S .

Example 2.4.1 We now consider several important structures that have played significant roles in logic, automata, and verification. The first structure is naturals with addition $\langle \mathbb{N}, + \rangle$. The addition relation $+$ is interpreted as a 3-ary relation consisting of tuples $(n, m, k) \in \mathbb{N}^3$ such that k is the sum of n and m . The second structure is nonnegative integers with linear order $\langle \mathbb{N}, < \rangle$, where the binary relation $<$ consists of pairs $(n, m) \in \mathbb{N}^2$ such that n is smaller than m . The third structure is naturals with successor $\langle \mathbb{N}, \text{succ} \rangle$, where the 2-ary relation succ consists of pairs $(n, m) \in \mathbb{N}^2$ with $m = n + 1$. Another important structure is $\langle \{0, 1\}^*, \text{succ}_0, \text{succ}_1 \rangle$, where the binary relation succ_i ($i = 0, 1$) contains pairs of words of the form (v, vi) with $v \in \{0, 1\}^*$. The

structure $\langle \{0, 1\}^*, \text{succ}_0, \text{succ}_1 \rangle$ can be naturally interpreted as the infinite complete binary tree with the root ε . We could also equip this structure with the transitive closure \preceq of $\text{succ}_0 \cup \text{succ}_1$ yielding the structure $\langle \{0, 1\}^*, \text{succ}_0, \text{succ}_1, \preceq \rangle$. Observe that, when $\langle \{0, 1\}^*, \text{succ}_0, \text{succ}_1 \rangle$ is viewed as the infinite binary tree, the relation \preceq can be interpreted as the descendant relation between nodes in the tree. ♣

In the sequel, we shall mostly deal with *transition systems*, which are defined as structures over vocabulary σ with only names of arity two. Therefore, transition systems are simply edge-labeled directed graphs. In the sequel, we shall often use the notation \rightarrow_a instead of R_a to denote the binary edge relation with name a , and write $s \rightarrow_a s'$ instead of $(s, s') \in \rightarrow_a$. Since we mostly use transition systems to model the evolution (or behavior) of some dynamic objects, the elements in the universe S of a transition system $\mathfrak{S} = \langle S, \{\rightarrow_a\}_{a \in \sigma} \rangle$ are called *configurations*². The edge relation \rightarrow_a will be called the *transition relation* labeled a , while the edges in \rightarrow_a are called *a-labeled transitions*. In this case, each name in σ is also said to be an *action label*. Hence, we will also use the notation ACT to denote the vocabulary σ and call ACT an *action alphabet*.

Example 2.4.2 Some examples of transition systems include the structures $\langle \mathbb{N}, \text{succ} \rangle$, $\langle \mathbb{N}, < \rangle$, $\langle \{0, 1\}^*, \text{succ}_0, \text{succ}_1 \rangle$, and $\langle \{0, 1\}^*, \text{succ}_0, \text{succ}_1, \preceq \rangle$ which we defined in Examples 2.4.1. Also, each NWA \mathcal{A} over Σ (omitting initial and final states) can be easily construed as transition systems over the action alphabet Σ . Another example of transition systems are those which are generated by a pushdown automaton $\mathcal{P} = (\Sigma, \Gamma, Q, \delta, q_0, F)$, i.e., containing configurations of \mathcal{P} as vertices, and transition relations \rightarrow_a for each $a \in \Sigma_e$. For more details of transition systems generated by pushdown automata, see Chapter 3. ♣

2.5 Logics and properties

In this section, we shall review the definitions of the logics and verification properties that we will consider in the sequel. For a more detailed treatment, the reader is referred to [BBF⁺01, BdRV01, CGP99, Lib04, Sti01, Tho96, vD08].

²they are often called *states* in the literature of modal logic, but we have reserved this term for automata

2.5.1 Safety, liveness, and fairness

We shall now define safety, liveness, and fairness properties, which are probably the most important properties in verification. Each of these is actually a *class* of properties, instead of a single property. Such properties are often treated informally in the literature since they are often definable in some temporal logics or in terms of some other properties including reachability and recurrent reachability (see subsequent subsections). We shall now recall the informal definitions of safety, liveness, and fairness properties. See [BBF⁺01, Chapters 6–11] for a more thorough treatment.

Two most commonly considered properties in verification are *safety* (under some conditions, no “bad” configurations are ever reachable) and *liveness* (under some conditions, some “good” configurations are eventually reachable). For example, the property that “the system never reaches a configuration where two processes are in a critical region” is an important safety property for mutual exclusion protocols, while the property that “every philosopher who requests noodle will eventually get it” is a liveness property that is often considered for protocols for dining philosopher problems [BA06, Lyn96]. To prove or disprove a safety property, it suffices to consider only finite executions of the systems since a violation of the property can be witnessed by a finite path that takes the system from an initial configuration to a bad configuration. On the other hand, this is not the case with liveness properties. To prove or disprove a liveness property, we need to take into account (potentially infinite) *maximal* executions of the systems.

Fairness is another important property in verification. Roughly speaking, it states that under certain conditions some events must occur infinitely often. One important use of fairness property is that liveness property in a system can often be reduced to a fairness property in a modified system via an automata-theoretic technique (see below). Another use of fairness property is as a *hypothesis* of a liveness property. Liveness property is often easily violated unless some fairness hypothesis is imposed. For example, for a dining philosopher protocol with n philosophers, an “unfair” path in the system could simply ignore a philosopher’s request indefinitely and therefore resulting in a violation of a desired liveness property. One common fairness hypothesis in the setting of distributed protocols is that if a resource is requested infinitely, it must be infinitely often granted.

2.5.2 Reachability and recurrent reachability

Safety, liveness, and fairness are often best expressed in terms of reachability and recurrent reachability. We shall now define these concepts. Given a transition system $\mathfrak{S} = \langle S, \{\rightarrow_a\}_{a \in \text{ACT}} \rangle$ and a subset $\text{ACT}' \subseteq \text{ACT}$, we write $\rightarrow_{\text{ACT}'}$ to denote the relation $\left(\bigcup_{a \in \text{ACT}'} \rightarrow_a \right)$ and \rightarrow to denote the relation \rightarrow_{ACT} . As usual, for a binary relation $R \subseteq S \times S$, we write R^+ (resp. R^*) for the transitive (resp. transitive-reflexive) closure of R . Given $s, t \in S$, we say that s can *reach* t in \mathfrak{S} (or t is *reachable* from s) if it is the case that $s \rightarrow^* t$. In the sequel, the *reachability relation* for the system \mathfrak{S} is the relation \rightarrow^* , while its *one-step reachability relation* is the relation \rightarrow . We shall also call \rightarrow^+ the *strict reachability relation* for the system \mathfrak{S} . It is also convenient to refer to the preimages and postimages of a relation. To this end, given a set $S' \subseteq S$ of configurations and a relation $R \subseteq S \times S$, we write

$$\begin{aligned} \text{pre}(S')[R] &:= \{v \in S : \exists w \in S' ((v, w) \in R)\}, \\ \text{post}(S')[R] &:= \{v \in S : \exists w \in S' ((w, v) \in R)\}. \end{aligned}$$

Given a transition system $\mathfrak{S} = \langle S, \{\rightarrow_a\}_{a \in \text{ACT}} \rangle$ and a subset $S' \subseteq S$, we now define the following sets:

$$\begin{aligned} \text{pre}(S') &:= \text{pre}(S')[\rightarrow], \\ \text{post}(S') &:= \text{post}(S')[\rightarrow], \\ \text{pre}^*(S') &:= \text{pre}(S')[\rightarrow^*], \\ \text{post}^*(S') &:= \text{post}(S')[\rightarrow^*], \\ \text{pre}^+(S') &:= \text{pre}(S')[\rightarrow^+], \\ \text{post}^+(S') &:= \text{post}(S')[\rightarrow^+]. \end{aligned}$$

We now move to recurrent reachability. Given a set S , a subset $S' \subseteq S$, and a relation $R \subseteq S \times S$, we denote by $\text{Rec}(S')[R]$ the set of all elements $s_0 \in S$ for which there exists an infinite sequence $\{s_i\}_{i \in \mathbb{Z}_{\geq 1}}$ such that:

- $s_i \in S'$ for all $i \in \mathbb{Z}_{\geq 1}$, and
- $(s_j, s_i) \in R$ for all pairs of distinct integers satisfying $0 \leq j < i$.

See Figure 2.6 for an illustration of an infinite sequence witnessing $s_0 \in \text{Rec}(S')[R]$. Given a transition system $\mathfrak{S} = \langle S, \{\rightarrow_a\}_{a \in \text{ACT}} \rangle$ and a subset $S' \subseteq S$, we use the nota-

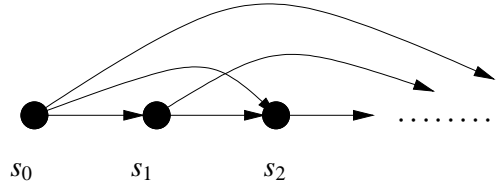


Figure 2.6: An illustration of an infinite sequence witnessing $s_0 \in \text{Rec}(S')[R]$. The edges are from the relation R , while each configuration s_i (with $i > 0$) must belong to S' . The configurations in this sequence are not necessarily all different.

tion $\text{Rec}(S')$ to denote $\text{Rec}(S')[\rightarrow^+]$. In this case, by transitivity of \rightarrow^+ we have $s_0 \in S'$ iff there exists an infinite sequence $\{s_i\}_{i \in \mathbb{Z}_{\geq 1}}$ such that $s_i \in S'$ and $s_{i-1} \rightarrow^+ s_i$ for all $i \in \mathbb{Z}_{\geq 1}$. In other words, the set $\text{Rec}(S')$ is the set of configurations in S' from which there exists an infinite path visiting S' infinitely often.

2.5.3 FO: First-order logic

We assume basic knowledge of mathematical logic (cf. [Lib04, vD08]). We shall only briefly recall some basic definitions and results on first-order logic.

Syntax and semantics

Let VAR be a countable set of (first-order) variables. In the sequel, we shall use x_i, y_i, z_i for variables with $i \in \mathbb{N}$. The syntax of first-order formulas over the vocabulary σ can be defined inductively as follows: (i) $x = y$ is an atomic formula for not necessarily distinct variables x and y (ii) if $a \in \sigma$ and x_1, \dots, x_n are (not necessarily distinct) variables, where $n = \text{AR}(a)$, then $R_a(x_1, \dots, x_n)$ is an (atomic) formula, (iii) if ϕ and ψ are formulas, then so are their conjunction $\phi \wedge \psi$, their disjunction $\phi \vee \psi$, and the negation $\neg\phi$, and (iv) if ϕ is a formula, then so are $\exists x\phi$ and $\forall x\phi$. Notice that we allow the standard built-in equality relation '='. The formula ϕ is said to be *existential positive* if it is of the form $\exists x_1, \dots, x_n \psi$, where ψ is *quantifier-free*, i.e., a boolean combinations of atomic formulas. The *free variables* $\text{free}(\phi)$ of a first-order formula ϕ are also built inductively: (i) $\text{free}(x = y) = \{x, y\}$ (ii) $\text{free}(R_a(x_1, \dots, x_{\text{AR}(a)})) = \{x_1, \dots, x_{\text{AR}(a)}\}$, (iii) $\text{free}(\phi \vee \psi) = \text{free}(\phi \wedge \psi) = \text{free}(\phi) \cup \text{free}(\psi)$, (iv) $\text{free}(\neg\phi) = \text{free}(\phi)$, and (iv) $\text{free}(\exists x\phi) = \text{free}(\forall x\phi) = \text{free}(\phi) \setminus \{x\}$. If ϕ is a formula with free variables x_1, \dots, x_n , then we write $\phi(x_1, \dots, x_n)$ to emphasize which free variables the formula ϕ has. A first-order *sentence* is simply a first-order formula ϕ with $\text{free}(\phi) = \emptyset$.

Fix a σ -structure $\mathfrak{S} = \langle S, \{R_a\}_{a \in \sigma} \rangle$. A \mathfrak{S} -*valuation* v is a function mapping the set VAR of variables to elements of S . Whenever \mathfrak{S} is clear from the context, we shall simply say *valuation*. If ϕ is a formula over σ , we may define the notion of *truth* of ϕ in \mathfrak{S} with respect to v in the standard way (cf. [Lib04, vD08]). If ϕ is true in \mathfrak{S} with respect to v , then we also say that \mathfrak{S} *satisfies* ϕ with respect to v and write $\mathfrak{S} \models \phi[v]$. A standard proposition in mathematical logic is that the truth of $\phi(x_1, \dots, x_n)$ in \mathfrak{S} only depends on the values of the valuation v on the free variables x_1, \dots, x_n of ϕ : $\mathfrak{S} \models \phi[v]$ iff, for every valuation v' that agrees with v on x_1, \dots, x_n , it is the case that $\mathfrak{S} \models \phi[v']$. For this reason, we shall often simply write $\mathfrak{S} \models \phi(v(x_1), \dots, v(x_n))$ whenever $\mathfrak{S} \models \phi[v]$ for some valuation v .

For two formulas $\phi(x_1, \dots, x_n)$ and $\psi(x_1, \dots, x_n)$ over the vocabulary σ with the same free variables, we say that ϕ and ψ are (*logically*) *equivalent*, written $\phi \equiv \psi$, if for every σ -structure \mathfrak{S} and valuation v it is the case that $\mathfrak{S} \models \phi[v]$ iff $\mathfrak{S} \models \psi[v]$. The following are several basic results on equivalence of first-order formulas: (i) $\neg\neg\phi \equiv \phi$, (ii) $\phi \wedge \psi \equiv \neg(\neg\phi \vee \neg\psi)$, (iii) $\phi \vee \psi \equiv \neg(\neg\phi \wedge \neg\psi)$, (iv) $\exists x\phi \equiv \neg\forall x\neg\phi$, and (v) $\forall x\phi \equiv \neg\exists x\neg\phi$. Therefore, we will sometimes assume that first-order formulas use only the operators $\{\vee, \neg, \exists\}$. Similarly, by pushing all the negations inside as much as possible, we may assume that the only occurrences of negations in first-order formulas are on the atomic level. For other standard equivalences, the reader is referred to [vD08].

Quantifier rank and alternation rank

The *quantifier rank* $qr(\phi)$ of a first-order formula ϕ is defined to be the maximum quantifier nesting depth inside ϕ . More formally, this notion can be defined inductively: (i) $qr(R_a(x_1, \dots, x_{AR(a)})) = qr(x = y) := 0$, (ii) $qr(\phi \vee \psi) := \max(qr(\phi), qr(\psi))$, (iii) $qr(\neg\phi) := qr(\phi)$, and (iv) $qr(\exists x\phi) := qr(\phi) + 1$.

Given a formula ϕ , let us push all the negations in ϕ to the atomic level. The *alternation rank* $AL(\phi)$ of a formula ϕ is defined to be the maximum number of alternations of operators in $\{\forall, \wedge\}$ and operators in $\{\exists, \vee\}$ over all paths from the root to the leaves in the parse tree of ϕ .

First-order queries

The somewhat non-standard notion of first-order queries that we shall next define is motivated by the standard notion of conjunctive queries in database theory literature

(cf. [AHV95]). A *first-order k -ary query* over the vocabulary σ is of the form

$$\mathbf{v}(x_1, \dots, x_k) \leftarrow \varphi(y_1, \dots, y_r)$$

where φ is a formula over σ , y_1, \dots, y_r are distinct variables, and x_1, \dots, x_k are *not* necessarily distinct variables. Given a σ -structure \mathfrak{S} , we define the *image* of \mathfrak{S} under \mathbf{v} to be the set

$$[[\mathbf{v}]]_{\mathfrak{S}} := \{(\mathbf{v}(x_1), \dots, \mathbf{v}(x_k)) : \mathbf{v} \text{ is a } \mathfrak{S}\text{-valuation s.t. } \mathfrak{S} \models \varphi[\mathbf{v}]\}.$$

We shall call $\mathbf{v}(x_1, \dots, x_k)$ the *head* of the query \mathbf{v} , while $\varphi(y_1, \dots, y_r)$ is called the *body* of the query \mathbf{v} . Without loss of generality, since first-order formulas are closed under existential quantification, we may assume that the variables y_1, \dots, y_r in the body of \mathbf{v} are among variables x_1, \dots, x_k in the head of the query.

Albeit it is the case that first-order queries can be crudely dealt with using only first-order formulas, there are two main reasons for using the notion of first-order queries. Firstly, the definition of first-order queries enforces an *explicit* ordering of the arguments in the induced k -ary relations, which is important when using automata to recognize relations (e.g. see Proposition 3.1.1). Secondly, unlike first-order formulas, the arity of the relations defined by first-order queries need not coincide with the number of free variables in their bodies. More importantly, we shall see later that first-order queries provide a cleaner notation for proofs.

FO^k : restriction of FO to k variables

In the literature of model theory (e.g. see [Lib04]), it is common to restrict the expressive power of FO by enforcing only k variables in the formulas, for a fixed $k \in \mathbb{Z}_{\geq 1}$. In the sequel, we use FO^k to denote the k -variable first-order logic containing the set of all first-order formulas which only use at most k variables.

First-order logic with two or three variables are often already sufficiently powerful. One well-known example is the equivalence between FO^3 over finite words and star-free regular expressions due to McNaughton and Papert [MP71]. We shall now make this statement more precise only over the alphabet $\{0, 1\}$, although it generalizes to any alphabet. A finite word $w = a_1 \dots a_n$ over the alphabet $\{0, 1\}$ can be thought of as a finite structure $\mathfrak{S} = \langle S, <, U \rangle$, where

- $S = \{1, \dots, n\}$,
- $<$ is the standard transitive binary ordering relation over $\{1, \dots, n\}$, and

- $U = \{i \in S : a_i = 1\}$ is a unary relation.

For example, the word 011 corresponds to the structure $\langle \{1, 2, 3\}, <, U \rangle$, where $<$ is the standard less-than relation over $\{1, 2, 3\}$ and $U = \{2, 3\}$. The reverse interpretation can similarly be done. Therefore, given a first-order sentence ϕ over the vocabulary consisting of a binary relation name $<$ and a unary relation name U , we may define $\mathcal{L}(\phi)$ to be the class of finite words w over $\{0, 1\}$ such that $w \models \phi$. We say that a language $\mathcal{L} \subseteq \{0, 1\}^*$ is *definable* in FO (resp. FO^k) if there exists a formula in FO (resp. FO^k) such that $\mathcal{L}(\phi) = \mathcal{L}$. McNaughton and Papert [MP71] proved that a regular language is star-free iff it is definable in FO. Other proofs for this result can also be found in [Lib04, Tho96]. In fact, it is folklore that FO^3 suffices and the translation to FO^3 from star-free regular expressions takes polynomial-time (e.g. see [EVW02] for a sketch).

Proposition 2.5.1 (McNaughton and Papert [MP71]) *A language is star-free regular iff it is definable in FO^3 . Furthermore, the translation from star-free regular expressions to FO^3 sentences can be performed in polynomial time.*

Many results in the literature show that a large number of modal and temporal logics can be embedded in first-order logic with two or three variables possibly extended with the transitive closure operators (cf. [BdRV01, EVW02, IK89, Kam68, Lib04]). We shall mention some of these results below.

2.5.4 $\text{FO}_{\text{REG}}(\text{Reach})$: FO with regular reachability

As we saw earlier, most properties in verification are related to the reachability property in some way. Therefore, a minimum criterion for a suitable logic in verification is that it needs to be able to express reachability. It is a well-known fact in model theory that first-order logic is not powerful enough to express reachability (cf. [Lib04]) over graphs. One way to overcome this limitation is to extend the logic with a reachability operator. We shall now define the logic $\text{FO}_{\text{REG}}(\text{Reach})$ that extends FO with “regular” reachability operators, and its subclass $\text{FO}(\text{Reach})$ which extends FO with the simplest reachability operators. These logics are natural and commonly considered in the context of verification (e.g. see [Col02, DT90, Löd03, LS05b, WT07]).

Given a finite set ACT of action labels, $\text{FO}_{\text{REG}}(\text{Reach})$ are built from atomic formulas of the form $x = y$, $x \rightarrow_a y$ ($a \in \text{ACT}$) and $\text{Reach}_{\mathcal{A}}(x, y)$ for an NWA \mathcal{A} over ACT , which we then close under boolean combinations and first-order quantifications using

the standard rules for first-order logic. The semantics for $\text{FO}(\text{Reach})$ are defined with respect to transition systems. They can be defined in the same way as for FO , except for formulas of the form $\text{Reach}_{\mathcal{A}}(x, y)$, which can be defined as follows: given a transition system \mathfrak{S} over ACT and a \mathfrak{S} -valuation v , we define that $\mathfrak{S} \models \text{Reach}_{\mathcal{A}}(v(x), v(y))$ iff there exists a path

$$s_0 \xrightarrow{a_1} \dots \xrightarrow{a_n} s_n$$

in \mathfrak{S} such that $s_0 = v(x)$, $s_n = v(y)$, and $a_1 \dots a_n \in \mathcal{L}(\mathcal{A})$. In other words, $\mathfrak{S} \models \text{Reach}_{\mathcal{A}}(v(x), v(y))$ iff the configuration $v(x)$ can reach $v(y)$ via a sequence of actions that are permitted by the NWA \mathcal{A} .

We define the logic $\text{FO}(\text{Reach})$ to be the sublogic of $\text{FO}_{\text{REG}}(\text{Reach})$ where we only permit atomic formulas of the form $\text{Reach}_{\mathcal{A}}(x, y)$, where $\mathcal{L}(\mathcal{A}) = \Gamma^*$ for some $\Gamma \subseteq \text{ACT}$. In the sequel, we shall use the shorthand $\text{Reach}_{\Gamma}(x, y)$ to refer to such an atomic formula, and the shorthand $\text{Reach}(x, y)$ to denote $\text{Reach}_{\text{ACT}}(x, y)$. The logic $\text{FO}_{\text{REG}}(\text{Reach})$ is probably the weakest extension of FO that can express reachability in a meaningful way.

The notion of quantifier rank can be easily extended to $\text{FO}_{\text{REG}}(\text{Reach})$ from FO by interpreting $\text{qr}(\text{Reach}_{\mathcal{A}}(x, y)) := 0$. The same goes with the notion of alternation rank. As before, we use $\text{FO}_{\text{REG}}^k(\text{Reach})$ (resp. $\text{FO}^k(\text{Reach})$) to denote the restrictions of $\text{FO}_{\text{REG}}(\text{Reach})$ (resp. $\text{FO}(\text{Reach})$) to formulas with at most k variables.

2.5.5 HM-logic: Hennessy-Milner Logic

It is well-known that many properties in verification cannot distinguish bisimulation-invariant properties. Due to its intimate connection with the notion of bisimulation, Hennessy-Milner logic³ plays an important role in verification. We shall now briefly recall the definition of Hennessy-Milner logic, the notion of bisimulations, and several basic results that are relevant to this thesis; for a more thorough treatment, the reader is referred to [BdRV01, Lib04, Sti01]. *Hennessy-Milner logic (HM-logic)* over the action alphabet ACT is defined by the following grammar:

$$\phi, \psi := \top \mid \neg\phi \mid \phi \vee \psi \mid \langle \text{ACT}' \rangle \phi \quad (\text{ACT}' \subseteq \text{ACT}).$$

If $a \in \text{ACT}$, then we write $\langle a \rangle \phi$ to denote $\langle \{a\} \rangle \phi$. We also use the usual abbreviations $\perp := \neg\top$, $\phi \wedge \psi := \neg(\neg\phi \vee \neg\psi)$, and $[\text{ACT}']\phi := \neg\langle \text{ACT}' \rangle \neg\phi$. The semantics $\llbracket \phi \rrbracket_{\mathfrak{S}}$

³Hennessy-Milner logic (modulo minor syntactic issues) is just *modal logic*, which was much earlier introduced in philosophy (cf. [BdRV01]).

of an HM-logic formula φ with respect to a transition system $\mathfrak{S} = \langle S, \{\rightarrow_a\}_{a \in \text{ACT}} \rangle$ is simply a subset of S defined inductively as follows:

- $\llbracket \top \rrbracket_{\mathfrak{S}} := S$,
- $\llbracket \neg \varphi \rrbracket_{\mathfrak{S}} := S \setminus \llbracket \varphi \rrbracket_{\mathfrak{S}}$,
- $\llbracket \varphi \vee \psi \rrbracket_{\mathfrak{S}} := \llbracket \varphi \rrbracket_{\mathfrak{S}} \cup \llbracket \psi \rrbracket_{\mathfrak{S}}$, and
- $\llbracket \langle \text{ACT}' \rangle \varphi \rrbracket_{\mathfrak{S}} := \{s \in S : \exists s' \in S (s \rightarrow_{\text{ACT}'} s' \text{ and } s' \in \llbracket \varphi \rrbracket_{\mathfrak{S}})\}$.

Given a configuration $s \in S$, we also write $\mathfrak{S}, s \models \varphi$ iff $s \in \llbracket \varphi \rrbracket_{\mathfrak{S}}$. The *problem of model checking HM-logic* is defined as follows: given a system $\mathfrak{S} = \langle S, \{\rightarrow_a\}_{a \in \text{ACT}} \rangle$, a configuration $s \in S$, and an HM-logic formula φ over ACT , decide whether $\mathfrak{S}, s \models \varphi$.

Standard translation to FO^2

It is well-known that HM-logic formulas can be thought of as first-order formulas with two variables. More precisely, for every HM-logic formula φ over ACT , there exists an FO formula $\varphi'(x)$ over ACT with one free variable such that, for every transition system $\mathfrak{S} = \langle S, \{\rightarrow_a\}_{a \in \text{ACT}} \rangle$ and $s \in S$, it is the case that

$$\mathfrak{S}, s \models \varphi \quad \Leftrightarrow \quad \mathfrak{S} \models \varphi'(s).$$

Furthermore, there exists an algorithm that performs this translation in linear time (cf. [BdRV01, Lib04]).

2.5.6 EF_{REG} -logic: HM-logic with regular reachability

HM-logic is not capable of expressing reachability. For this reason, we introduce EF_{REG} -logic, which is HM-logic with a regular reachability operator, and its syntactic restriction EF-logic, which is HM-logic with a simple reachability operator. More precisely, the syntax of EF_{REG} -logic over the action alphabet ACT is the extension of the syntax of HM-logic over ACT with the following rule:

- if \mathcal{A} is an NWA over ACT and φ an EF_{REG} -logic formula over ACT , then $\text{EF}_{\mathcal{A}}\varphi$ is an EF_{REG} -logic formula over ACT .

The semantics of formula of the form $\text{EF}_{\mathcal{A}}\varphi$ is defined as follows:

- for a transition system $\mathfrak{S} = \langle S, \{\rightarrow_a\}_{a \in \text{ACT}} \rangle$ and $s_0 \in S$, we have $\mathfrak{S}, s_0 \models \text{EF}_{\mathcal{A}}\varphi$ iff there exists a path

$$s_0 \rightarrow_{a_1} \dots \rightarrow_{a_n} s_n$$

in \mathfrak{S} such that $a_1 \dots a_n \in \mathcal{L}(\mathcal{A})$ and $s_n \in \llbracket \varphi \rrbracket_{\mathfrak{S}}$.

We then define $\llbracket \text{EF}_{\mathcal{A}}\varphi \rrbracket_{\mathfrak{S}}$ to be the set of configurations $s_0 \in S$ such that $\mathfrak{S}, s_0 \models \text{EF}_{\mathcal{A}}\varphi$. We define EF-logic to be the restriction of EF_{REG} -logic which allows only reachability operators of the form $\text{EF}_{\mathcal{A}}$, where $\mathcal{L}(\mathcal{A}) = \Gamma^*$ for some $\Gamma \subseteq \text{ACT}$. In the sequel, we shall use the shorthand EF_{Γ} to refer to such a reachability operator, and the shorthand EF to denote EF_{ACT} .

Remark 2.5.1 In the verification literature, EF-logic is often defined in such a way that only the reachability operator EF is allowed (e.g. see [BEM97, LS02, May98, Wal00]). However, it is known that permitting the more general operator EF_{Γ} does not change the complexity of model checking problems in most cases. For this reason, we shall adopt the more general definition. ■

We saw earlier that HM-logic can be naturally thought of as a fragment FO^2 . In the same manner, EF_{REG} -logic can be thought of as a fragment of $\text{FO}_{\text{REG}}^2(\text{Reach})$ and EF-logic a fragment of $\text{FO}^2(\text{Reach})$. Furthermore, the same linear-time translation can be used in this case.

2.5.7 CTL: Computation Tree logic

CTL is one of the most common branching-time temporal logics that are considered in the context of verification (cf. [BBF⁺01, CGP99, Lib04, Sti01]). Loosely speaking, it is an extension of EF-logic with powerful “constraints on the way”. To be more precise, let us define the syntax and semantics of CTL. The syntax of the logic CTL over the action alphabet ACT is the extension of the syntax of EF-logic with the following two rules:

- if φ and ψ are CTL formulas over ACT, then $\text{E}(\varphi \cup \psi)$ is also a CTL formula
- if φ is a CTL formula over ACT, then so is $\text{EG}\varphi$.

The semantics of these types of formulas are defined with respect to a transition system $\mathfrak{S} = \langle S, \{\rightarrow_a\}_{a \in \text{ACT}} \rangle$ and a configuration $s_0 \in S$ as follows:

- $\mathfrak{S}, s_0 \models E(\varphi \cup \psi)$ iff there exists a path

$$s_0 \rightarrow \dots \rightarrow s_n$$

such that $\mathfrak{S}, s_n \models \psi$ and, for each $i = 0, \dots, n-1$, we have $\mathfrak{S}, s_i \models \varphi$.

- $\mathfrak{S}, s_0 \models EG(\varphi)$ iff there exists a (finite or infinite) maximal path $s_0 \rightarrow s_1 \rightarrow \dots$ such that $\mathfrak{S}, s_i \models \varphi$, for each $i \in \mathbb{N}$.

As before, we use $\llbracket \varphi \rrbracket_{\mathfrak{S}}$ to denote the set of configurations $s \in S$ such that $\mathfrak{S}, s \models \varphi$.

The *problem of CTL model checking* can be defined as follows: given a finite system $\mathfrak{S} = \langle S, \{\rightarrow_a\}_{a \in \text{ACT}} \rangle$, a configuration $s \in S$, and a CTL formula φ over ACT, decide whether $\mathfrak{S}, s \models \varphi$. It is well-known (cf. [CGP99]) that model checking CTL over finite systems can be done in polynomial time. This definition can also be easily extended to finitely representable infinite-state systems by permitting finite representations of \mathfrak{S} and s to be the input.

2.5.8 LTL: Linear Temporal Logic

Linear Temporal Logic (LTL) is one of the most standard and natural temporal logics considered in the context of verification. It has been argued that the logic is more intuitive than branching-time temporal logics like CTL (cf. [Var01]). We shall now recall the definition of LTL and review some of its most important results. See [Var95, Wol00] for a more thorough treatment.

The syntax of LTL over ACT is defined as follows:

$$\varphi, \varphi' := a \ (a \in \text{ACT}) \mid \neg\varphi \mid \varphi \vee \varphi' \mid \varphi \wedge \varphi' \mid \mathbf{X}\varphi \mid \varphi \mathbf{U}\varphi'.$$

We shall use the standard abbreviations: $\mathbf{F}\varphi$ for $\text{true}\mathbf{U}\varphi$, $\mathbf{G}\varphi$ for $\neg\mathbf{F}\neg\varphi$, and \mathbf{F}_s and \mathbf{G}_s for their strict versions: $\mathbf{F}_s\varphi = \mathbf{X}\mathbf{F}\varphi$ and $\mathbf{G}_s\varphi = \neg\mathbf{F}_s\neg\varphi$. The semantics of LTL over ACT is given by ω -word languages over ACT:

- $\llbracket a \rrbracket = \{v \in \text{ACT}^\omega : v(1) = a\},$
- $\llbracket \neg\varphi \rrbracket = \text{ACT}^\omega - \llbracket \varphi \rrbracket,$
- $\llbracket \varphi \vee \varphi' \rrbracket = \llbracket \varphi \rrbracket \cup \llbracket \varphi' \rrbracket,$
- $\llbracket \varphi \wedge \varphi' \rrbracket = \llbracket \varphi \rrbracket \cap \llbracket \varphi' \rrbracket,$
- $\llbracket \mathbf{X}\varphi \rrbracket = \{v \in \text{ACT}^\omega : v[1, \infty) \in \llbracket \varphi \rrbracket\},$ and

- $\llbracket \phi \mathbf{U} \phi' \rrbracket = \{v \in \text{ACT}^\omega : \exists j \geq 0 \forall i < j (v[i, \infty) \in \llbracket \phi \rrbracket \wedge v[j, \infty) \in \llbracket \phi' \rrbracket)\}$.

Given an ω -word $v \in \text{ACT}^\omega$ and an LTL formula ϕ over ACT , we write $v \models \phi$ iff $v \in \llbracket \phi \rrbracket$. We shall now recall a seminal result by Vardi and Wolper [VW86a].

Proposition 2.5.2 (Vardi-Wolper [VW86a]) *Given an LTL formula ϕ over ACT , we can compute an NBWA \mathcal{A}_ϕ of size $2^{O(\|\phi\|)}$ such that $\mathcal{L}(\mathcal{A}_\phi) = \llbracket \phi \rrbracket$ in time $2^{O(\|\phi\|)}$.*

Given a transition system $\mathfrak{S} = \langle S, \{\rightarrow_a\}_{a \in \text{ACT}} \rangle$ and a word $v = a_1 a_2 \dots \in \text{ACT}^\omega$, we say that $s_0 \in S$ *realizes* v if there is an infinite path

$$s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} s_2 \xrightarrow{a_3} \dots$$

in \mathfrak{S} . We define the semantics of LTL over transition systems in the standard way: $(\mathfrak{S}, v) \models \phi$ iff every ω -word $v \in \text{ACT}^\omega$ realized by v satisfies ϕ . We write $\llbracket \phi \rrbracket_{\mathfrak{S}}^\forall$ for the set of all $v \in S$ such that $(\mathfrak{S}, v) \models \phi$. Here, the symbol \forall is used to signify the universal semantics that is adopted in the definition (i.e. *every* path starting in v satisfies ϕ). Dually, we will write $\llbracket \phi \rrbracket_{\mathfrak{S}}^\exists$ for the complement of the set $\llbracket \neg \phi \rrbracket_{\mathfrak{S}}^\forall$, i.e., for the set of $v \in S$ from which *there exists* a path that satisfies ϕ .

The *problem of LTL model checking* can be defined as follows: given a finite system $\mathfrak{S} = \langle S, \{\rightarrow_a\}_{a \in \text{ACT}} \rangle$, a configuration $s \in S$, and an LTL formula, decide whether $\mathfrak{S}, s \models \phi$. This definition can be easily extended to finitely representable infinite-state systems by allowing a finite representation of \mathfrak{S} and s as the inputs. In the case of finite systems, the problem is known to be PSPACE-complete, which can be shown using a model-theoretic technique [SC85] or an automata-theoretic technique (i.e. a more refined version of Proposition 2.5.2) [Var95, VW86a, Wol00]. See the survey [Sch02] for a more thorough discussion.

2.5.9 Other logics

There are of course other logics that are common in verification. We shall mention a few others, but not give a precise definition since we will not encounter them in the sequel. Among others, we mention:

- *Monadic second-order logic (MSO)*. This is simply first-order logic extended with quantification over sets of elements (see [Lib04, Tho96, Tho03] for a definition). This logic is perhaps the most expressive logic in verification, i.e., it subsumes virtually all logics that are considered in verification.

- *μ -calculus*. This is simply modal logic extended with least fixed point operators (see [Sti01, Lib04] for a definition). This logic is subsumed in MSO, but subsumes most logics that are invariant under bisimulation.
- *Propositional dynamic logic (PDL)*. This is simply modal logic extended with recursion and other modalities (see [BdRV01] for a definition). There are a number of variations of this logic with different expressive power, but the most basic PDL is subsumed in μ -calculus.

2.5.10 Model checking complexity

When we deal with the problem of model checking with respect to a certain logic L , there are usually several parameters that are considered as part of the problems. The two most important ones are: (1) structures, and (2) formulas.

Let us first clarify how we measure the size of structures and formulas. In this thesis, when structures are part of the model checking problems, they always take shape of transition systems, i.e., edge-labeled directed graphs. In the case of finite systems, we may measure them in the standard way we measure the size of graphs. When they are *implicitly* (or *symbolically*) represented (e.g. when the systems are infinite), we will simply use the size of these symbolic representations, which we will define when defining a symbolic language for the representations. In the case of formulas, we will simply measure the size of the parse trees of the given formula (except for Chapter 8 when we represent our formulas as dags). In the case of logic which allows natural numbers as constants (e.g. Presburger Arithmetic with syntactic sugar), we will also measure the numbers in binary, unless stated otherwise.

There are three standard complexity measures when dealing with model checking depending on which of the two input parameters are fixed. The first measure is *combined complexity* when both structures and formulas are considered as part of the input. This measure is reasonable when both structures and formulas are important parameters. The second measure is *data complexity* which considers only structures to be part of the input, i.e., formulas are fixed. This measure is considered to be the most useful measure in the context of verification since for most applications the size of the structures can be extremely large, while only small formulas are used in practice. The third measure is *expression complexity* which considers only formulas to be part of the input, i.e., the structures are fixed. Many believe that the third measure is not very reasonable in practice. This is, however, not true since many verification problems these days

are achieved by reductions to Presburger Arithmetic, S1S, or S2S. Nowadays there are fast solvers that have been developed for these theories (e.g. MONA [HJJ⁺95], LIRA [BDEK07], and Omega [Ome]).

Part I

Generic Approaches: Algorithmic Metatheorems

Chapter 3

Word/Tree-automatic Systems

Generic approaches to infinite-state model checking require generic frameworks that are expressive enough to subsume Turing-powerful models of computation. Many such frameworks have been proposed in the literature (cf. [AJNS04, Bar07, BGR10, BFLS05, BFLP08, BG09, BLN07, Blu99, BG04, Boi99, Bou01, BLW03, BLW09, BW94, BJNT00, BHV04, DLS02, FL02, KMM⁺97, KMM⁺01, JN00, Mor00, Nil05, Rub08, WB98]). Such frameworks often make use finite state automata (or equivalent models) as finite representations of the transition relations and the domains of transition systems in various ways. Although the use of finite state automata often yield nice closure and algorithmic properties, they are not in general sufficient for the verification of reachability or more complex properties due to expressive power of the framework. In this thesis, we adopt *word automatic systems* [Blu99, BG04] and *tree automatic systems* [Blu99, BG04, BLN07] as our generic frameworks since they strike a good balance between the expressive power (e.g. they subsume many decidable classes of infinite-state transitions systems) and closure/algorithmic properties (e.g. effective closure under boolean combinations and automata projections).

Our purpose of using generic frameworks in this thesis significantly differs from common uses of generic frameworks in the literature of infinite-state model checking. For example, generic frameworks are often used in combination with semi-algorithms for computing reachability sets and reachability relations (cf. [AJNS04, BFLS05, BFLP08, Boi99, BLW03, Bou01, BW94, BJNT00, BHV04, BLW09, DLS02, Nil05, KMM⁺97, KMM⁺01, JN00, WB98]). Although the results in this thesis can be used in conjunction with such semi-algorithms, we shall chiefly use generic frameworks as frameworks for deriving *algorithmic metatheorems for decidable model checking*, which are generic results that can be used in a “plug-and-play” manner for inferring de-

cidability of certain model checking tasks over *a large family* of formalisms of infinite-state systems, instead of doing so for *a single* formalism at a time. Such a use of generic frameworks is not new, e.g., this can be found in the work of [BFLP08, BFLS05, LS04, LS05a] on flattable linear counter systems, which derives a single semi-algorithm for reachability that is *guaranteed* to terminate over many important subclasses of Petri nets and counter systems (cf. [LS04, LS05a]). We shall give our algorithmic metatheorems for word/tree automatic transition systems in the next two chapters of the thesis.

This chapter aims to review basic definitions and results for word/tree automatic transition systems [Blu99, BG04], and compare them with several other closely-related generic frameworks that have been considered in the literature. The chapter is organized as follows. We review the definition and standard results of word automatic transition systems in Section 3.1, and of tree automatic systems in Section 3.2. In particular, we shall review basic properties of word/tree automatic transition systems and give several well-known concrete classes of infinite-state systems that they can capture. In Section 3.3, we shall discuss a number of other well-known generic frameworks that have been considered in the literature — in particular, length-preserving word-automatic systems, rational transition systems, Presburger-definable transition systems, and ω -automatic transition systems — and compare them with word/tree automatic systems in terms of expressive power and closure/algorithmic properties.

3.1 Word-automatic systems

In this section, we shall define the framework of word-automatic systems [Blu99, BG04]. The reader is also referred to [Bar07, BGR10, Rub08] for more recent results regarding automatic structures.

3.1.1 Basic definitions

Loosely speaking, word-automatic systems $\mathfrak{S} = \langle S, \{\rightarrow_a\}_{a \in \text{ACT}} \rangle$ are those transition systems such that, for some alphabet Σ , S is a regular language over Σ and \rightarrow_a is a “regular relation” over Σ . To define the notion of regular relations, we need a binary operation over Σ^* called *convolution* \otimes , which computes an encoding of a pair of words over Σ^* as a word over some new alphabet. More precisely, given words $v, w \in \Sigma^*$ where $v = a_1 \dots a_n$ and $w = b_1 \dots b_m$, let $v \otimes w$ be the word $c_1 \dots c_k$ over the alphabet

$\Sigma_{\perp} \times \Sigma_{\perp}$, where $\Sigma_{\perp} := \Sigma \cup \{\perp\}$ with $\perp \notin \Sigma$, $k = \max(n, m)$, and

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

Roughly speaking, the word $v \otimes w$ is obtained by putting v on top of w and padding the shorter word by the padding symbol \perp . For example, when $v = aab$ and $w = ababab$, the word $v \otimes w$ is simply

$$\begin{bmatrix} a \\ a \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} \begin{bmatrix} b \\ a \end{bmatrix} \begin{bmatrix} \perp \\ b \end{bmatrix} \begin{bmatrix} \perp \\ a \end{bmatrix} \begin{bmatrix} \perp \\ b \end{bmatrix}.$$

A relation $\rightarrow_a \subseteq \Sigma^* \times \Sigma^*$ is said to be *regular* if the language $\{v \otimes w : v \rightarrow_a w\}$ is regular.

Example 3.1.1 The relation $= \subseteq \{0, 1\}^* \times \{0, 1\}^*$ consisting of pairs of equal words (u, v) is obviously regular, e.g., it is generated by the regular expression $\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \end{bmatrix} \right)^*$. The relation $\mathbf{el} \subseteq \{0, 1\}^* \times \{0, 1\}^*$ consisting of words $(u, v) \in \{0, 1\}^* \times \{0, 1\}^*$ of equal length is regular, e.g., it is generated by the regular expression $(\{0, 1\} \times \{0, 1\})^*$. The prefix-of relation $\preceq \subseteq \{0, 1\}^* \times \{0, 1\}^*$ consisting of words (u, uw) for some words $u, w \in \{0, 1\}^*$ is also regular, e.g., it is generated by the regular expression $\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \end{bmatrix} \right)^* (\{\perp\} \times \{0, 1\})^*$. ♣

In the sequel, we shall not distinguish a relation and its language representation. Let us now summarize the definition of word-automatic systems as follows.

Definition 3.1.1 A transition system $\mathfrak{S} = \langle S, \{\rightarrow_a\}_{a \in \text{ACT}} \rangle$ is said to be Σ^* -automatic if S is a regular language over Σ and each \rightarrow_a is a regular relation over Σ . It is said to be word-automatic (or just automatic) if it is Σ^* -automatic for some alphabet Σ .

As there are multiple ways of representing a given regular language, there are also non-unique ways of representing a given word-automatic system. This motivates the following definition. A *presentation* of a Σ^* -automatic system $\langle S, \{\rightarrow_a\}_{a \in \text{ACT}} \rangle$ is a tuple $\eta = \langle \mathcal{A}_S, \{\mathcal{A}_a\}_{a \in \text{ACT}} \rangle$ where \mathcal{A}_S and \mathcal{A}_a 's are NWAs such that $\mathcal{L}(\mathcal{A}_S) = S$ and $\mathcal{L}(\mathcal{A}_a) = \{v \otimes w : v \rightarrow_a w\}$ for each $a \in \text{ACT}$. We shall also use the notation \mathfrak{S}_η to denote the transition system of which η is a presentation. A transition system \mathfrak{S}' over ACT is said to be (word-)automatically presentable if it is isomorphic to some automatic system \mathfrak{S} over ACT.

Remark 3.1.1 Two remarks are in order. Firstly, our definition of automatic systems is slightly different from the common definition in the literature of automatic structures (e.g. see [BG04]); the latter coincides with our definition of automatically presentable systems. Nonetheless, both approaches are equivalent since we are only interested in verification problems, instead of the expressive power of certain logics, over automatic systems. Secondly, our definition of automatic presentations use NWAs as default representations of regular languages. One may of course adopt other representations of regular languages, such as regular expressions or DWAs. Nonetheless, our choice of representations of regular languages is justified by computational complexity reasons, i.e., that most systems that are considered in the literature permit succinct automatic presentations in terms of NWAs (but not necessarily in terms of DWAs or regular languages), while the complexity of the algorithmic metatheorems that we obtain in the thesis do not increase even if we adopt NWAs (instead of DWAs or regular languages) as default representations of regular languages.

3.1.2 Examples

We now give four examples of classes of infinite-state systems which can be construed as word-automatic systems; more will be given in subsequent chapters.

Example 3.1.2 (Pushdown systems) A *pushdown system* (PDS) is a PDA without an initial state and the set of final states (cf. [BEM97, May98, MS85, Tho03]). This omission is due to the fact that we are no longer interested in PDAs as acceptors of languages, but instead as generators of infinite transition systems. More precisely, given a PDA $(\Sigma, \Gamma, Q, \delta, q_0, F)$, the tuple $\mathcal{P} := (\text{ACT}, \Gamma, Q, \delta)$ is a PDS over the action alphabet ACT, which we define to be the set of elements a in Σ_ϵ for which there exists a transition rule in δ of the form $((q, a, u), (q', u'))$. Many notions for PDAs (e.g. configurations) can be easily adapted to PDSs. The PDS \mathcal{P} gives rise to the transition system $\mathfrak{S}_{\mathcal{P}} = \langle S, \{\rightarrow_a\}_{a \in \text{ACT}} \rangle$, where $S \subseteq Q \times \Gamma^*$ is the set of configurations of \mathcal{P} and $\rightarrow_a \subseteq S \times S$ is the binary relation containing tuples $((q, vu), (q', vu'))$ such that there exists a transition $((q, a, u), (q', u')) \in \delta$. A simple example of a transition system generated by a PDS (up to isomorphism) is the structure S2S; see Example 2.4.1 for a definition. This can, in fact, be generated by a PDS with one state.

Transition systems generated by pushdown systems can be easily thought of as word-automatic systems as follows. Given a PDS $\mathcal{P} = (\text{ACT}, \Gamma, Q, \delta)$, let $\Omega = Q \cup \Gamma$.

We define the set S' and, for each $a \in \text{ACT}$, the relation \rightarrow'_a as follows:

$$\begin{aligned} S' &:= Q\Gamma^* \\ \rightarrow'_a &:= \{(qu, q'u') \in S' \times S' : (q, u) \rightarrow_a (q', u')\}. \end{aligned}$$

In other words, by interpreting each configuration (q, u) of \mathcal{P} as the word $qu \in Q\Gamma^*$, we see that the transition system $\mathfrak{S}'_{\mathcal{P}} = \langle S', \{\rightarrow'_a\}_{a \in \text{ACT}} \rangle$ is isomorphic to $\mathfrak{S}_{\mathcal{P}}$. Furthermore, the isomorphism function can be implemented to run efficiently, i.e., in linear time. It is now not hard to see that $\mathfrak{S}'_{\mathcal{P}}$ is word-automatic for which a presentation can be computed in time $O(|Q \cup \Gamma|^2 + |\Gamma| \times \|\mathcal{P}\|)$. This is because each NWA \mathcal{A}_a for \rightarrow'_a is over the alphabet Ω_{\perp}^2 and behaves as follows:

1. nondeterministically guess a transition in $(q, a, u, q', u') \in \delta$,
2. make sure that $\begin{bmatrix} q \\ q' \end{bmatrix}$ is the first letter read,
3. read a word of the form $v \otimes v \in (\Gamma \times \Gamma)^*$, and
4. nondeterministically jump to a state to check that the rest of the input word is $u \otimes u'$.

In other words, these two representations of pushdown systems are polynomially equivalent. Therefore, in the sequel we shall use the term “pushdown system” to refer to either of these representations. ♣

Example 3.1.3 (Prefix-recognizable systems) Prefix-recognizable systems are natural generalizations of pushdown systems, where we allow potentially infinitely many rules which are represented using regular languages (cf. [Cau03]). More precisely, a *prefix-recognizable system*¹ \mathcal{P} over ACT is a tuple $(\text{ACT}, \Gamma, Q, \delta)$ where

- Γ is a finite stack alphabet,
- Q is a finite set of states, and
- δ is a transition relation, i.e., a finite set of transitions of the form

$$((q, a, \mathcal{A}), (q', \mathcal{A}'), \mathcal{A}''),$$

where $q, q' \in Q$, $a \in \text{ACT}$, and $\mathcal{A}, \mathcal{A}', \mathcal{A}''$ are NWAs over Γ .

¹In the literature, prefix-recognizable systems are usually defined without state components. However, the two definitions are easily seen to be equivalent.

As for PDSs, a *configuration* is simply a pair $(q, w) \in Q \times \Gamma^*$ consisting of a state and a word in Γ^* . Given two configurations (q, w) and (q', w') of \mathcal{P} , we write $(q, w) \rightarrow_a (q', w')$ if there exist three words $\alpha, \beta, \gamma \in \Gamma^*$ and a rule $((q, a, \mathcal{A}), (q', \mathcal{A}'), \mathcal{A}'')$ in δ such that $w = \alpha\beta$, $w' = \alpha\gamma$, $\alpha \in \mathcal{L}(\mathcal{A}'')$, $\beta \in \mathcal{L}(\mathcal{A})$, and $\gamma \in \mathcal{L}(\mathcal{A}')$. The transition system $\mathfrak{S}_{\mathcal{P}}$ generated by \mathcal{P} is simply the system $\langle S, \{\rightarrow_a\}_{a \in \text{ACT}} \rangle$, where $S \subseteq Q \times \Gamma^*$ is simply the set of all configurations of \mathcal{P} , and \rightarrow_a is the one-step reachability relation via action a that we just defined. To understand the definition of prefix-recognizable systems and the transition systems they generate, it is helpful to draw an analogy with pushdown systems. We may think of pushdown systems as prefix-recognizable systems $\mathcal{P} = (\Sigma, \Gamma, Q, \delta)$, where each rule in δ is of the form $((q, a, \mathcal{A}), (q', \mathcal{A}'), \mathcal{A}'')$ for some NWAs \mathcal{A} and \mathcal{A}' that accepts only a single word in Γ^* and for some \mathcal{A}'' that accepts all words in Γ^* . A simple example of a transition system that can easily be generated by a prefix-recognizable system (up to isomorphisms) is $\langle \{0, 1\}^*, \text{succ}_0, \text{succ}_1, \preceq \rangle$ (see Example 2.4.1). This, however, cannot be generated by pushdown systems since the nodes in $\langle \{0, 1\}^*, \text{succ}_0, \text{succ}_1 \rangle$ have an infinite degree.

Transition systems generated by prefix-recognizable systems can be easily thought of as word-automatic systems as follows. Given a prefix-recognizable system $\mathcal{P} = (\text{ACT}, \Gamma, Q, \delta)$, let $\Omega := Q \cup \Gamma$. We define a transition system $\mathfrak{S}'_{\mathcal{P}} = \langle Q\Gamma^*, \{\rightarrow'_a\}_{a \in \text{ACT}} \rangle$ in the same way as in the previous example. In other words, we interpret each configuration (q, u) of \mathcal{P} as the word $qu \in Q\Gamma^*$. It is easy to see that the new transition system $\mathfrak{S}'_{\mathcal{P}}$ is isomorphic to $\mathfrak{S}_{\mathcal{P}}$. Furthermore, the isomorphism function can be computed in linear time. Similarly, it is not hard to see that $\mathfrak{S}'_{\mathcal{P}}$ is automatic for which a presentation can be computed in time $O(|Q \cup \Gamma|^2 + \|\mathcal{P}\|)$. The construction for the NWAs for each \rightarrow'_a is a simple adaptation of the construction in the previous example. In the sequel, when the meaning is clear from the context, we shall use the term “prefix-recognizable system” to refer to either of these representations of prefix-recognizable systems. ♣

Example 3.1.4 Counter machines [Min67] are a well-known Turing-powerful model of computation. We shall now define the notion of counter systems, which are simply counter machines without initial and final states. A k -counter system \mathcal{M} over the action alphabet ACT is a tuple $(\text{ACT}, X, Q, \Delta)$ where

- X is a set of k (*counter*) *variables*, say $\{x_1, \dots, x_k\}$,
- Q is a set of *states*,
- Δ is a finite set of *instructions* of the form $((q, \phi(X)), a, (q', i_1, \dots, i_k))$, where:

- $q, q' \in Q$,
- $a \in \text{ACT}$,
- each i_j is a number in $\{-1, 0, 1\}$
- $\varphi(X)$ is a (guard) Presburger formula of the form $\bigwedge_{x \in Y} x \sim_x 0$ for some $Y \subseteq X$ and $\sim_x \in \{=, >\}$.

A *configuration* of \mathcal{M} is a tuple $(q, n_1, \dots, n_k) \in Q \times \mathbb{N}^k$ expressing the state \mathcal{M} is in and the current values of the k counters. Given two configurations $\mathbf{c}_1 = (q, n_1, \dots, n_k)$ and $\mathbf{c}_2 = (q', n'_1, \dots, n'_k)$, we write $\mathbf{c}_1 \rightarrow_a \mathbf{c}_2$ if there exists an instruction

$$((q, \varphi(X)), a, (q', i_1, \dots, i_k))$$

such that the guard formula $\varphi(n_1, \dots, n_k)$ is true in $\langle \mathbb{N}, + \rangle$, and for each $j = 1, \dots, k$ we have $n_j = \max(0, n_j + i_j)$. In other words, when the counter value is 0, it stays 0 when \mathcal{M} tries to subtract 1 from it. The transition system generated by \mathcal{M} is simply $\mathfrak{S}_{\mathcal{M}} := \langle S, \{\rightarrow_a\}_{a \in \text{ACT}} \rangle$, where $S \subseteq Q \times \mathbb{N}^k$ is the set of all configurations of \mathcal{M} and \rightarrow_a is the one-step reachability relation via action a that we just defined. It is well-known that the reachability problem for counter systems (i.e. checking whether a given configuration c_2 of a counter system \mathcal{M} is reachable in $\mathfrak{S}_{\mathcal{M}}$ from another given configuration c_1 of \mathcal{M}) is undecidable [Min67].

We can think of transition systems generated by k -counter systems \mathcal{M} as automatic systems in two different ways depending on whether we adopt the standard binary representation of numbers (cf. [Kla08, WB00]), or its reverse (cf. [BC96, BHMV94]). In the sequel, we shall adopt the reversed binary representation of numbers from [BC96, BHMV94]. More precisely, given a number n , let **rev-bin**(n) denote the standard binary representation of n (with unnecessary leading 0s removed) written in *reverse* order, e.g., **rev-bin**(8) = 0001. Given the numbers $n_1, \dots, n_k \in \mathbb{N}$, define $n_1 \otimes^0 \dots \otimes^0 n_k$ as the word **rev-bin**(n_1) $\otimes \dots \otimes$ **rev-bin**(n_k) with the symbol \perp replaced by the symbol 0. For example, $7 \otimes^0 8 \otimes^0 6$ is the word

$$\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}.$$

Therefore, $n_1 \otimes^0 \dots \otimes^0 n_k$ is a word over the alphabet $\{0, 1\}^k$. Define a new alphabet $\Omega := Q \cup \{0, 1\}^k$. Then, we can define a function χ mapping a given configuration

$\mathbf{c} = (q, n_1, \dots, n_k) \in Q \times \mathbb{N}^k$ of \mathcal{M} to the word $qw \in \Omega^*$, where $w = n_1 \otimes^0 \dots \otimes^0 n_k$. We may therefore think of the set of configurations of \mathcal{M} as the set

$$S := \{\chi(\mathbf{c}) : \mathbf{c} \in Q \times \mathbb{N}^k\}$$

and the relation \rightarrow_a as the relation

$$\rightarrow'_a := \{(\chi(\mathbf{c}), \chi(\mathbf{c}')) : \mathbf{c} \rightarrow_a \mathbf{c}'\}.$$

It is easy to construct an NWA \mathcal{A}_S for the set S with 2 states in time $O(|Q| + 2^k)$, most of which is spent in enumerating the letters in the alphabet Ω . Similarly, it is not hard to construct in time $O(|Q|^2 \times 2^k)$ an NWA \mathcal{A}_a with $O(\|\mathcal{M}\| \times 2^k)$ states over Ω_\perp^2 for \rightarrow'_a . Intuitively, the automaton first nondeterministically guesses a transition in \mathcal{M} that will be executed and remember it in its finite memory. Upon reading any input letter $\{0, 1\}^k \times \{0, 1\}^k$, it will remember in its finite memory precisely one carry bit for each of the k counters. ♣

Example 3.1.5 Given a Turing machine $\mathcal{M} = (\Sigma, \Gamma, Q, \delta, q_0, q_F, \square)$, define the transition system $\mathfrak{S}_{\mathcal{M}} = \langle S, \rightarrow \rangle$, where $S = \Gamma^*(Q \times \Gamma)\Gamma^*$ is the set of all configurations of \mathcal{M} and \rightarrow is the one-step reachability relation defined by \mathcal{M} . The reachability problem for transition systems generated by Turing machines is well-known to be undecidable.

It is known that transition systems generated by Turing machines are automatic [BG04]. Define $\Omega := \Gamma \cup (Q \times \Gamma)$. The set S of configurations of \mathcal{M} is therefore regular. We can also easily construct an NWA for the relation $\rightarrow \subseteq S \times S$ since \mathcal{M} makes only local changes at each step (i.e. at most three cells and the state of \mathcal{M}). In fact, this automatic presentation for $\mathfrak{S}_{\mathcal{M}}$ can be computed in time polynomial in $\|\mathcal{M}\|$.

♣

3.1.3 Basic closure and algorithmic results

Given $v_1, \dots, v_n \in \Sigma^*$, let us write $v_i = a_{i,1} \dots a_{i,j_i}$ for each $i = 1, \dots, n$. Letting $m = \max(j_1, \dots, j_n)$, we define $v_1 \otimes \dots \otimes v_n$ as the word $c_1 \dots c_m$ over the alphabet Σ_\perp^n where, for each $k = 1, \dots, m$, $c_k := (c_{1,k}, \dots, c_{n,k})$ and

$$c_{i,k} := \begin{cases} a_{i,k} & \text{if } k \leq j_i \\ \perp & \text{if } j_i < k \leq m. \end{cases}$$

An r -ary relation $R \subseteq (\Sigma^*)^r$ is said to be *regular* if the language

$$\{v_1 \otimes \dots \otimes v_r : (v_1, \dots, v_r) \in R\}$$

is regular. Observe that this definition generalizes our earlier definition of binary regular relations. As before, we do not distinguish a relation and its language representation.

Definition 3.1.2 A Σ^* -automatic structure over the vocabulary σ is a σ -structure $\mathfrak{S} = \langle S, \{R_a\}_{a \in \sigma} \rangle$ where S is a regular language over Σ and R_a is an $\text{AR}(a)$ -ary regular relation over S . A σ -structure is said to be automatic if it is Σ^* -automatic for some alphabet Σ .

An automatic presentation η of a Σ^* -automatic structure $\mathfrak{S}_\eta = \langle S, \{R_a\}_{a \in \sigma} \rangle$ is a tuple $\langle \mathcal{A}_S, \{\mathcal{A}_a\}_{a \in \sigma} \rangle$, where \mathcal{A}_S is an NWA over Σ with $\mathcal{L}(\mathcal{A}_S) = S$ and \mathcal{A}_a is an NWA over $\Sigma_{\perp}^{\text{AR}(a)}$ such that $\mathcal{L}(\mathcal{A}_a) = R_a$. A σ -structure is said to be (word-)automatically presentable if it is isomorphic to an automatic structure over σ . Remark 3.1.1 for our definition of automatic systems also holds for our definition of automatic structures.

Example 3.1.6 Presburger arithmetic $\langle \mathbb{N}, + \rangle$ is well-known to be $\{0, 1\}^*$ -automatic (via the reverse binary encoding of numbers; see Example 3.1.4). In fact, the extension $\langle \mathbb{N}, +, |_2 \rangle$ with the binary relation $|_2$ is automatic, where for all $n, m \in \mathbb{N}$, $n |_2 m$ iff n divides m and $n = 2^k$ for some $k \in \mathbb{N}$. The structure $\langle \mathbb{N}, +, |_2 \rangle$ is also known as *Büchi Arithmetic*. See [BGR10, Blu99, BHMV94] for more details. The structure $\langle \{0, 1\}^*, \text{succ}_0, \text{succ}_1, \preceq \rangle = \langle \{0, 1\}^*, \text{succ}_0, \text{succ}_1, \preceq \rangle$ is also word-automatic [Blu99, BG04]. In fact, it is still automatic even when extended with an equal-length binary relation **el** defined in Example 3.1.1. For more examples, see the recent survey [BGR10]. ♣

The following closure properties are well-known.

Proposition 3.1.1 ([Hod83]) Given an automatic presentation η of an automatic structure $\mathfrak{S}_\eta = \langle S, \{R_a\}_{a \in \sigma} \rangle$ over the vocabulary σ and a first-order query $\mathfrak{v}(\bar{x}) \leftarrow \phi(\bar{y})$ over σ , the relation $[[\mathfrak{v}]]_{\mathfrak{S}_\eta}$ is effectively regular.

We shall next sketch a standard proof of this proposition in some details as it will be used in the sequel as a backbone of a more complex construction.

Proof Sketch. We shall adopt NWAs as representations of regular languages. Suppose that $\eta = \langle \mathcal{A}_S, \{\mathcal{A}_a\}_{a \in \text{ACT}} \rangle$. Inductively on the structure of the query body $\phi(\bar{y})$, we shall construct an NWA over Σ_{\perp}^m , where $m = \text{AR}(\mathfrak{v})$, such that for all $v_1, \dots, v_m \in S$, it is the case that

$$(v_1, \dots, v_m) \in [[\mathfrak{v}]]_{\mathfrak{S}_\eta} \quad \Leftrightarrow \quad v_1 \otimes \dots \otimes v_m \in \mathcal{L}(\mathcal{A}_\phi).$$

An NWA for $\llbracket \mathbf{v} \rrbracket_{\mathfrak{S}_\eta}$ can later be obtained by taking a product of \mathcal{A}_ϕ with the regular set $\mathcal{L}_m := \underbrace{S \otimes \dots \otimes S}_{m \text{ times}}$, for which an NWA of size $\|\mathcal{A}_S\|^m$ can be easily constructed.

- Base case: $\phi := R_a(x_{i_1}, \dots, x_{i_r})$ for an r -ary relation R_a and some (not necessarily distinct) indices $1 \leq i_1, \dots, i_r \leq m$. If $\mathcal{A}_a = (\Sigma_\perp^r, Q, \delta, Q_0, F)$ is the NWA for R_a in the presentation η , then we construct the NWA $\mathcal{A}_\phi = (\Sigma_\perp^m, Q, \delta', Q_0, F)$ where

$$\delta'(q, (a_1, \dots, a_m)) := \delta(q, (a_{i_1}, \dots, a_{i_r})).$$

Therefore, we have $\mathcal{L}(\mathcal{A}_\phi) \cap \mathcal{L}_m = \llbracket \mathbf{v} \rrbracket$. Note that $\|\mathcal{A}_\phi\| = \|\mathcal{A}_a\|$. The time taken to construct \mathcal{A}_ϕ is clearly. $O(|\Sigma|^m \times \|\mathcal{A}_a\|)$.

- Inductive case: $\phi := \neg\psi(x_{i_1}, \dots, x_{i_r})$ for some indices $1 \leq i_1, \dots, i_r \leq m$. Define the new query

$$\mathbf{v}'(x_1, \dots, x_m) \leftarrow \psi(x_{i_1}, \dots, x_{i_r}).$$

Let $\mathcal{A}_{\mathbf{v}'}$ be an NWA such that $\mathcal{L}(\mathcal{A}_{\mathbf{v}'}) \cap \mathcal{L}_m = \llbracket \mathbf{v}' \rrbracket$ which can be obtained by induction. To obtain an NWA \mathcal{A}_ϕ such that $\mathcal{L}(\mathcal{A}_\phi) \cap \mathcal{L}_m = \llbracket \mathbf{v} \rrbracket$, we determinize and then complement $\mathcal{A}_{\mathbf{v}'}$. The number $|\mathbf{States}(\mathcal{A}_\phi)|$ of states of \mathcal{A}_ϕ is at most exponential in the number $|\mathbf{States}(\mathcal{A}_{\mathbf{v}'})|$ of states of $\mathcal{A}_{\mathbf{v}'}$. It follows that $\|\mathcal{A}_\phi\| \leq 2^{2|\mathbf{States}(\mathcal{A}_{\mathbf{v}'})|}$. The time taken to construct \mathcal{A}_ϕ is $2^{O(|\mathbf{States}(\mathcal{A}_{\mathbf{v}'})| + m \log(|\Sigma|))}$ on top of the time taken to construct $\mathcal{A}_{\mathbf{v}'}$.

- Inductive case: $\phi := \phi'(x_{i_1}, \dots, x_{i_s}) \vee \phi''(x_{j_1}, \dots, x_{j_r})$ for some indices

$$1 \leq x_{i_1}, \dots, x_{i_s} \leq m \quad \text{and} \quad 1 \leq j_1, \dots, j_r \leq m.$$

Define new queries

$$\mathbf{v}'(x_1, \dots, x_m) \leftarrow \phi'(x_{i_1}, \dots, x_{i_s})$$

and

$$\mathbf{v}''(x_1, \dots, x_m) \leftarrow \phi''(x_{j_1}, \dots, x_{j_r}).$$

By induction, we can construct the NWAs $\mathcal{A}_{\mathbf{v}'}$ and $\mathcal{A}_{\mathbf{v}''}$ such that

$$\mathcal{L}(\mathcal{A}_{\mathbf{v}'}) \cap \mathcal{L}_m = \llbracket \mathbf{v}' \rrbracket,$$

$$\mathcal{L}(\mathcal{A}_{\mathbf{v}''}) \cap \mathcal{L}_m = \llbracket \mathbf{v}'' \rrbracket.$$

To obtain an NWA \mathcal{A}_ϕ such that $\mathcal{L}(\mathcal{A}_\phi) \cap \mathcal{L}_m = \llbracket \mathbf{v} \rrbracket$, we simply perform NWA union of $\mathcal{A}_{\mathbf{v}'}$ and $\mathcal{A}_{\mathbf{v}''}$. It follows that $\|\mathcal{A}_\phi\| = \|\mathcal{A}_{\mathbf{v}'}\| + \|\mathcal{A}_{\mathbf{v}''}\|$. The time taken to construct \mathcal{A}_ϕ is $O(|\Sigma|^m \times (\|\mathcal{A}_{\mathbf{v}'}\| + \|\mathcal{A}_{\mathbf{v}''}\|))$ on top of the time taken to construct $\mathcal{A}_{\mathbf{v}'}$ and $\mathcal{A}_{\mathbf{v}''}$.

- Inductive case: $\varphi := \varphi'(x_{i_1}, \dots, x_{i_s}) \wedge \varphi''(x_{j_1}, \dots, x_{j_r})$ for some indices

$$1 \leq x_{i_1}, \dots, x_{i_s} \leq m \quad \text{and} \quad 1 \leq j_1, \dots, j_r \leq m.$$

This case is identical to the disjunction case, but instead we compute the product automata. The number of states of the resulting NWA is the product (instead of sum) of the number of states of the two NWAs obtained from induction. Such is also the case for the time taken to compute the NWA for $\llbracket \mathbf{v} \rrbracket$.

- Inductive case: $\varphi := \exists y \varphi'(x_{i_1}, \dots, x_{i_r}, y)$ for some indices $1 \leq x_{i_1}, \dots, x_{i_r} \leq m$. By induction, we can construct an NWA $\mathcal{A}_{\mathbf{v}'}$ such that $\mathcal{L}(\mathcal{A}_{\mathbf{v}'}) \cap \mathcal{L}_{m+1} = \llbracket \mathbf{v}' \rrbracket$, where \mathbf{v}' is defined as

$$\mathbf{v}'(x_1, \dots, x_m, y) \leftarrow \varphi'(x_{i_1}, \dots, x_{i_r}, y).$$

Observe that $\llbracket \mathbf{v} \rrbracket = \{(v_1, \dots, v_m) : \exists u \in S((v_1, \dots, v_m, u) \in \llbracket \mathbf{v}' \rrbracket)\}$. Therefore, we need to construct a new NWA $\mathcal{A}'_{\mathbf{v}'}$ from $\mathcal{A}_{\mathbf{v}'}$ in such a way that $\mathcal{L}(\mathcal{A}'_{\mathbf{v}'}) \cap (\mathcal{L}_m \times \Sigma_{\perp}^*) = \llbracket \mathbf{v}' \rrbracket$. More precisely, suppose that $\mathcal{A}_S = (\Sigma, Q_2, \delta_2, Q_0^2, F_2)$. Let $\mathcal{A}'_S = (\Sigma^{m+1}, Q^2, \Delta, Q_0^2, F^2)$ be the NWA such that $(q, (a_1, \dots, a_n, a_{n+1}), q') \in \Delta$ iff $(q, a_{n+1}, q') \in \delta_2$. Taking a product of $\mathcal{A}_{\mathbf{v}'}$ and \mathcal{A}'_S , we obtain an NWA $\mathcal{A}'_{\mathbf{v}'} = (\Sigma_{\perp}^{m+1}, Q', \delta', Q'_0, F')$ such that $\mathcal{L}(\mathcal{A}'_{\mathbf{v}'}) \cap (\mathcal{L}_m \times \Sigma_{\perp}^*) = \llbracket \mathbf{v}' \rrbracket$. We now shall construct an NWA $\mathcal{A}_{\mathbf{v}} = (\Sigma_{\perp}^m, Q, \delta, Q_0, F)$ such that $\mathcal{L}(\mathcal{A}_{\mathbf{v}}) \cap \mathcal{L}_m = \llbracket \mathbf{v} \rrbracket$ as follows:

- $Q := Q'$,
- $Q_0 := q'_0$,
- for each $q \in Q$ and $a_1, \dots, a_m \in \Sigma_{\perp}$, let

$$\delta(q, (a_1, \dots, a_m)) := \bigcup_{a \in \Sigma_{\perp}} \delta'(q, (a_1, \dots, a_m, a)),$$

and

- let F be the set of states $q \in Q'$ from which there exists a path π in $\mathcal{A}_{\mathbf{v}'}$ on some word $w \in (\{\perp\}^m \times \Sigma)^*$ to some state in F .

Such a construction is commonly known as NWA *projection*. Obviously, we have $F \subseteq F'$ but they might not necessarily coincide. The reason for this definition of F is simply that, given $(v_1, \dots, v_m) \in \llbracket \mathbf{v} \rrbracket$, the word u such that

$$(v_1, \dots, v_m, u) \in \llbracket \mathbf{v}' \rrbracket$$

might have length greater than $\max(v_1, \dots, v_m)$. Furthermore, the set F can be computed by the standard algorithm for testing nonemptiness for NWAs, which requires only linear time. Note that $\mathcal{A}_v = \|\mathcal{A}_{v'}\| \times \|\mathcal{A}_S\|$. The time taken to construct \mathcal{A}_v is at most $O(|\Sigma|^{m+1} \times \|\mathcal{A}_{v'}\| \times \|\mathcal{A}_S\|)$ on top of the time taken to construct $\mathcal{A}_{v'}$.

This completes our proof of the proposition. \square

It is easy to see that the aforementioned construction runs in nonelementary time. Clearly, the most expensive operation in the construction is complementation, which yields a DWA of exponential size. In contrast, NWA projection takes only linear time. A more careful analysis of the aforementioned construction, however, reveals that the bottleneck of the complexity of the construction comes from the number of alternations between negations and existential quantifiers in the given first-order formula: the former turns an NWA into a DWA of exponential size (for which all boolean operations can be easily done), but the latter turns a DWA back into an NWA (for which complementation is expensive).

A better complexity can be obtained for the above proposition when restricting to simpler fragments of first-order logic. The most commonly used fragment in the sequel is the class of *conjunctive queries*, i.e., first-order queries $v(\bar{x}) \leftarrow \exists y_1, \dots, y_n \varphi$, where φ is simply a conjunction of atomic formulas. The following proposition can be obtained in a straightforward way by applying the proof of Proposition 3.1.1.

Proposition 3.1.2 *Let*

$$v(x_1, \dots, x_m) \leftarrow \exists y_1, \dots, y_n \varphi$$

be a conjunctive query over σ , where

$$\varphi = R_{a_1}(\bar{z}_1) \wedge \dots \wedge R_{a_k}(\bar{z}_k)$$

is a conjunction of atomic formulas, where $\bar{z}_i \subseteq \bar{x} \cup \bar{y}$ for each $i = 1, \dots, k$ and each variable in \bar{x} occurs in one of \bar{z}_i at least once. Given a presentation $\eta = \langle \mathcal{A}_S, \{\mathcal{A}_a\}_{a \in \sigma} \rangle$ of the Σ^ -automatic structure \mathfrak{S}_η , an NWA \mathcal{A}_v accepting $\llbracket v \rrbracket_{\mathfrak{S}_\eta}$ of size $O(\prod_{i=1}^k \|\mathcal{A}_i\|)$ can be computed in time $O(|\Sigma|^{m+n} \times \prod_{i=1}^k \|\mathcal{A}_i\|)$.*

Example 3.1.7 In this example, we show how the above Proposition can be used to prove that the reachability relation of a transition system is “efficiently” interdefinable with the strict reachability relation, provided that they are regular. Suppose that $\mathfrak{S}_\eta = \langle S, \{\rightarrow_a\}_{a \in \text{ACT}} \rangle$ is a Σ^* -automatic transition system over ACT, presented by the

presentation $\eta = \langle \{\mathcal{A}_S\}, \{\mathcal{A}_a\}_{a \in \text{ACT}} \rangle$. Since regular languages are closed under union, the relation $\rightarrow = (\bigcup_{a \in \text{ACT}} \rightarrow_a)$ is recognized by an NWA \mathcal{N} of size $\sum_{a \in \text{ACT}} \|\mathcal{A}_a\|$ and is computable in time $O(|\Sigma|^2 \times \sum_{a \in \text{ACT}} \|\mathcal{A}_a\|)$. As we shall see later, the reachability relation $\rightarrow^* = (\bigcup_{a \in \text{ACT}} \rightarrow_a)^*$ and the strict version \rightarrow^+ are in general not regular (in fact, not even recursive).

Suppose, however, that \rightarrow^* is regular and is recognized by the NWA \mathcal{R} . Observe now that Proposition 3.1.2 implies that the strict reachability relation \rightarrow^+ is also regular, for which an NWA \mathcal{R}' of size $O(\|\mathcal{N}\| \times \|\mathcal{R}\|)$ is computable in time $O(|\Sigma|^3 \times \|\mathcal{N}\| \times \|\mathcal{R}\|)$. This is because the relation \rightarrow^+ is definable in the new structure $\mathfrak{S}' = \langle S, \rightarrow, \rightarrow^* \rangle$ as follows: $x \rightarrow^+ y \Leftrightarrow \exists z (x \rightarrow z \wedge z \rightarrow^* y)$. Conversely, if \rightarrow^+ is regular and is recognized by the NWA \mathcal{R} , then so is the relation \rightarrow^* since \rightarrow^* is nothing but the union of \rightarrow and \rightarrow^+ . This also implies that an NWA for \rightarrow^* of size $O(\|\mathcal{N}\| + \|\mathcal{R}\|)$ is computable in time $O(|\Sigma|^2 \times (\|\mathcal{N}\| + \|\mathcal{R}\|))$. ♣

It turns out that the above proposition easily extends to the more general class of existential positive first-order formulas.

Proposition 3.1.3 *Let*

$$\varphi(y_1, \dots, y_m) = \exists x_1, \dots, x_n \psi(x_1, \dots, x_n, y_1, \dots, y_m)$$

be a first-order formula over the vocabulary σ , where ψ is a positive boolean combination of atomic propositions with h conjunctions. Given an automatic presentation η of an automatic structure \mathfrak{S}_η , an NWA \mathcal{A}_φ accepting $\llbracket \varphi \rrbracket_{\mathfrak{S}_\eta}$ is computable in time polynomial in $\|\eta\|$ and $\|\psi\|$, but exponential in h and $n + m$.

3.1.4 Negative results

It turns out that the nonelementary complexity of the construction above is unavoidable [BG04, Grä90], even when the input formula has no free variables.

Proposition 3.1.4 *There exists an automatic structure whose first-order theory has nonelementary complexity.*

A simple example of an automatic structure with nonelementary first-order theory is S2S with descendant [CH90]. Since transition systems of Turing machines are automatic, the following proposition due to [BG04] is immediate.

Proposition 3.1.5 ([BG04]) *The reachability problem for automatic transition systems is undecidable. In fact, it is Σ_1^0 -complete even for a fixed automatic transition system.*

In fact, the Σ_1^0 -hardness lower bound follows from the fact that the transition systems generated by counter machines and Turing machines are automatically presentable. The upper bound is owing to the fact that reachability for automatic transition systems has a finite witness that can be effectively checked. We now adapt the proof of Proposition 3.1.5 to show that for the problem of checking *recurrent reachability for automatic transition systems* is much harder: given a presentation η of an automatic system $\mathfrak{S}_\eta = \langle S, \{\rightarrow_a\}_{a \in \text{ACT}} \rangle$, an initial configuration $s_0 \in S$, and an NWA \mathcal{A} , decide whether $s_0 \in \text{Rec}(\mathcal{L}(\mathcal{A}))[\rightarrow^+]$.

Proposition 3.1.6 *The recurrent reachability problem for automatic transition systems is Σ_1^1 -complete.*

Proof. To prove Σ_1^1 -hardness, we establish a many-to-one reduction from the *recurrent state properties for nondeterministic Turing machines*: given an NTM

$$\mathcal{M} = (\Sigma, \Gamma, Q, \delta, q_0, q_F)$$

and a state $q \in Q$, check whether \mathcal{M} have an infinite computation path visiting the state q infinitely often. This problem is known to be Σ_1^1 -complete (e.g. see [Har86, Corollary 6.2]). Therefore, we may use the construction of automatic presentation η for the transition system $\mathfrak{S}_\mathcal{M} = \langle S, \rightarrow \rangle$ generated by \mathcal{M} from Example 3.1.5, which works as well for NTMs. Therefore, \mathcal{M} has an infinite computation path visiting q infinitely often iff $(q_0, \square) \in \text{Rec}(\Gamma^*(\{q\} \times \Gamma)\Gamma^*)[\rightarrow]$. This completes the reduction.

One way to prove membership in Σ_1^1 is to establish a many-to-one reduction to the problem of recurrent state properties for NTMs. More precisely, given a presentation $\eta = \langle \mathcal{A}_S, \{\rightarrow_a\}_{a \in \text{ACT}} \rangle$ for the Ω^* -automatic system $\mathfrak{S}_\eta = \langle S, \{\rightarrow_a\}_{a \in \text{ACT}} \rangle$, an initial configuration $v_0 \in S$, and an NWA \mathcal{A}_F over Ω for a set $F \subseteq S$, we construct the NTM \mathcal{M} that has a special state q that is visited iff a signalling bit b is turned “on”. The machine \mathcal{M} initially replaces the input word on the tape with v_0 , turns “off” the bit b , and begins “exploring” the transition system \mathfrak{S}_η from v_0 . At each stage of \mathcal{M} ’s computation, \mathcal{M} will remember on the tape a word $w \in \Omega^*$ that is reachable from v_0 , and a bit b signalling whether $w \in F$. If $w \in F$ is signalled, then \mathcal{M} will visit the state q and then turn off the bit b before resuming the exploration of \mathfrak{S}_η from

the currently remembered configuration w . If $w \notin F$, the NTM \mathcal{M} will resume the exploration of \mathfrak{S}_η without first visiting the state q . After this, the machine \mathcal{M} will nondeterministically write down a word $w' \in \Omega^*$ on the tape. Note that this step of \mathcal{M} might not terminate, but is not important as the resulting infinite computation path will not visit q infinitely often. In the case when \mathcal{M} terminates with w' fully written on the tape, \mathcal{M} checks whether $w \rightarrow w'$, which could be easily done since an NWA for \rightarrow can be easily constructed. If $w \rightarrow w'$, then \mathcal{M} will set $w := w'$, adjust the value of the bit b according to whether $w' \in L$, and continue to the next stage. If $w \not\rightarrow w'$, the \mathcal{M} simply enters a halting state. Finally, it is easy to check that $v_0 \in \text{Rec}(F)$ iff the NTM \mathcal{M} has an infinite computation path visiting q infinitely often on the empty input. \square

Let us briefly revisit the proof of this proposition. For the proof, it is important that the NWA \mathcal{A} in the input has an infinite language. In fact, if \mathcal{A} recognizes a finite language, by pigeonhole principle one of the configurations in $\mathcal{L}(\mathcal{A})$, say s , must be visited infinitely often. This means that there exists *finite witnesses* for positive instances (i.e. a path from the initial configuration s_0 to s , and a path from s to itself), and therefore is recursively enumerable.

3.2 Tree-automatic systems

In this section, we review the basic definitions and results for tree-automatic systems [BLN07, Blu99, BG04]. We refer the reader to [Bar07, BGR10] for a more up-to-date exposition of the subject.

3.2.1 Basic definitions

The notion of tree-automatic systems is to a large extent similar to word-automatic systems, except that we use NTAs instead of NWAs to represent the domain and the transition relations of the systems. To make this notion more precise, we define the convolution operation \otimes over $\text{TREE}_k(\Sigma)$ as follows: given two trees $T_1, T_2 \in \text{TREE}_k(\Sigma)$ with $T_1 = (D_1, \tau_1)$ and $T_2 = (D_2, \tau_2)$, let $T_1 \otimes T_2$ be the k -ary tree (D, τ) over the alphabet $\Sigma_\perp \times \Sigma_\perp$, where $\Sigma_\perp := \Sigma \cup \{\perp\}$ with $\perp \notin \Sigma$, such that $D = D_1 \cup D_2$ and $\tau(u) = (a_1, a_2)$ where $a_i = \tau_i(u)$ if $u \in D_i$, or else $a_i = \perp$. Observe that this is a simple generalization of the word case. Figure 3.1 illustrates how this operation works. A relation $R \subseteq \text{TREE}_k(\Sigma) \times \text{TREE}_k(\Sigma)$ is said to be *tree-regular* (or simply *regular*) if the language $\{T_1 \otimes T_2 : (T_1, T_2) \in R\}$ is tree-regular.

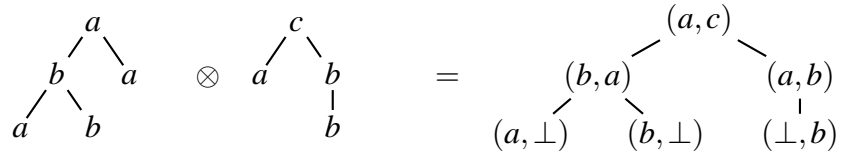


Figure 3.1: A specific example illustrating the convolution operation on binary trees

Example 3.2.1 The identity relation $=$ on $\text{TREE}_k(\Sigma)$ is obviously tree-regular. In fact, it can be recognized by an NTA with one state q , which only has transitions of the form $(q, (a, a), q, q, \dots, q)$ for $a \in \Sigma$. Similarly, the *equal tree-domain* relation \approx_{dom} on $\text{TREE}_k(\Sigma)$ (i.e. that two trees have the same tree domain but possibly different labelings) is also tree-regular, for which an NTA with one state could be easily constructed. The tree extension relation \preceq on $\text{TREE}_k(\Sigma)$ is also easily seen to be regular. In fact, one may construct an NTA for \preceq with precisely two states and at most $O(2|\Sigma|)$ transitions.



As in the case of word-automatic systems, we shall not distinguish a relation and its language representation. Let us now summarize the definition of tree-automatic systems.

Definition 3.2.1 A transition system $\mathfrak{S} = \langle S, \{\rightarrow_a\}_{a \in \text{ACT}} \rangle$ is said to be $\text{TREE}_k(\Sigma)$ -automatic if S is a regular tree language over $\text{TREE}_k(\Sigma)$ and each $\rightarrow_a \subseteq \text{TREE}_k(\Sigma) \times \text{TREE}_k(\Sigma)$ is a tree-regular relation. The system \mathfrak{S} is said to be tree-automatic if it is $\text{TREE}_k(\Sigma)$ -automatic for some k and Σ .

A presentation of a $\text{TREE}_k(\Sigma)$ -automatic system $\langle S, \{\rightarrow_a\}_{a \in \text{ACT}} \rangle$ is a tuple

$$\eta = \langle \mathcal{A}_S, \{\mathcal{A}_a\}_{a \in \text{ACT}} \rangle,$$

where \mathcal{A}_S is an NTA over $\text{TREE}_k(\Sigma)$ and each \mathcal{A}_a an NTA over $\text{TREE}_k(\Sigma) \times \text{TREE}_k(\Sigma)$, such that $\mathcal{L}(\mathcal{A}_S) = S$ and $\mathcal{L}(\mathcal{A}_a) = \{T \otimes T' : T \rightarrow_a T'\}$ for each $a \in \text{ACT}$. We shall use the notation \mathfrak{S}_η to denote the tree-automatic transition system of which η is a presentation. A transition system \mathfrak{S}' over ACT is said to be *tree-automatically presentable* if it is isomorphic to some tree-automatic system \mathfrak{S} over ACT . Clearly, the class of tree-automatic (resp. tree-automatically presentable) systems subsumes the class of word-automatic (resp. word-automatically presentable) systems.

Remark 3.2.1 Just as in the case of word-automatic systems, our definition of tree-automatic systems is slightly different from the common definition in the literature

of automatic structures (e.g. see [BG04]); the latter coincides with our definition of tree-automatically presentable systems. Nonetheless, both approaches are equivalent since we are only interested in verification problems, instead of the expressive power of certain logics. ■

3.2.2 Examples

We now give two examples of tree-automatic transition systems; more will be given in subsequent chapters.

Example 3.2.2 A ground tree rewrite system (GTRS) over ACT (cf. [DT90, Löd03]) is a tuple $\mathcal{P} = (k, \Sigma, \Delta)$, where

- k is a positive integer,
- Σ is a labeling alphabet, and
- Δ is a finite set of rewrite rules of the form (t, a, t') , where $t, t' \in \text{TREE}_k(\Sigma)$ and $a \in \text{ACT}$.

The GTRS \mathcal{P} gives rise to a transition system $\mathfrak{S}_{\mathcal{P}} = \langle S, \{\rightarrow_a\}_{a \in \text{ACT}} \rangle$ where $S := \text{TREE}_k(\Sigma)$ and, for all trees $T, T' \in \text{TREE}_k(\Sigma)$ where $T = (D, \tau)$, we have $T \rightarrow_a T'$ iff for some rewrite rule $(t, a, t') \in \mathcal{P}$ and $u \in D$ it is the case that t is a subtree of T rooted at u and $T' = T[t'/u]$. It is easy to see that ground tree rewrite systems generalize pushdown systems in their expressive power as generators of transition systems.

Transition systems generated by GTRSs can be easily construed as tree-automatic systems as follows. Given a GTRS $\mathcal{P} = (k, \Sigma, \Delta)$, let $\mathfrak{S}_{\mathcal{P}} = \langle S, \{\rightarrow_a\}_{a \in \text{ACT}} \rangle$ be the transition system generated by \mathcal{P} . It is easy to produce an NTA that recognizes each \rightarrow_a . More precisely, an NTA \mathcal{A}_a for \rightarrow_a can be either in a “guessing mode”, “idle mode”, or “checking mode”. Suppose $T \in \text{TREE}_k(\Sigma^2_{\perp})$ is the input tree to \mathcal{A}_a , and let $\mathbf{virt}(T) = (D, \tau)$ and $u \in D$. When \mathcal{A}_a is an idle mode, it simply ensures that the subtree rooted at u is an identity relation (i.e. the subtree rooted at u when projected onto the first component coincides with the subtree rooted at u when projected onto the second component). When \mathcal{A}_a is in guessing mode, it nondeterministically guesses (with ε -transitions) whether the current node is the root of the subtree where the rewriting takes place. When the guess is negative, \mathcal{A}_a chooses one of its children to “pass on” the guessing mode, while the rest of the children are to be in idle mode. When the guess is positive, it *instantaneously* switches to a checking mode. Then \mathcal{A}_a chooses a rewrite

rule $(t, a, t') \in \delta$ encoded in its finite memory and ensures that the subtree rooted at u when projected onto the first component coincides with t (minus the padding symbol), and when projected onto the second component coincides with t' (minus the padding symbol). This can be done by *simultaneously* verifying the two components. The final states are declared to be the union of the idle state and the states after successful checks have been made.

Let us now measure the computation time to produce a tree-automatic presentation for the transition systems generated by GTRSs. Analyzing the algorithm above, it is easy to see that the NTA over $\text{TREE}_k(\Sigma_\perp^2)$ that recognizes the one-step reachability is of size $O(\|\mathcal{P}\|)$ and can be computed in time $O(|\Sigma|^2 \times \|\mathcal{P}\|)$. In other words, these two representations of ground tree rewrite systems are polynomially equivalent. Therefore, in the sequel we shall use the term “ground tree rewrite systems” to refer to either of these representations. ♣

Example 3.2.3 We now present a generalization of ground tree rewrite systems called *regular ground tree rewrite systems (RGTRSs)* (cf. [DT90, Löd03, Löd06]). Intuitively, RGTRSs extend ground tree rewrite systems in the same way prefix-recognizable systems extend pushdown systems. More precisely, a *regular ground tree rewrite system (RGTRS)* over ACT is a tuple $\mathcal{P} = (k, \Sigma, \Delta)$, where k and Σ are the same as for GTRS and Δ is a finite set of rules of the form $(\mathcal{A}, a, \mathcal{A}')$, where \mathcal{A} and \mathcal{A}' are NTAs over $\text{TREE}_k(\Sigma)$ and $a \in \text{ACT}$. This GTRS \mathcal{P} gives rise to a transition system $\mathfrak{S}_{\mathcal{P}} = \langle S, \{\rightarrow_a\}_{a \in \text{ACT}} \rangle$ where $S = \text{TREE}_k(\Sigma)$ and, for all trees $T, T' \in \text{TREE}_k(\Sigma)$ where $T = (D, \tau)$, we have $T \rightarrow_a T'$ iff for some rewrite rule $(\mathcal{A}, a, \mathcal{A}') \in \Delta$, a tree $t \in \mathcal{L}(\mathcal{A})$, a tree $t' \in \mathcal{L}(\mathcal{A}')$, and a node $u \in D$ it is the case that t is a subtree of T rooted at u and $T' = T[t'/u]$. It is known (cf. [Löd03]) that RGTRSs subsume prefix-recognizable systems, although the latter could be exponentially more succinct than the former.

It is not difficult to see that transition systems generated by RGTRSs are tree-automatic. This can be proven in the same way as for GTRSs, e.g., for the verification stage, we can do product construction. It is easy to see that the resulting NTA over $\text{TREE}_k(\Sigma_\perp^2)$ is of size $O(\|\mathcal{P}\|^2)$ and can be computed in time $O(|\Sigma|^2 \times \|\mathcal{P}\|^2)$. In other words, these two representations of RGTRSs are polynomially equivalent. Therefore, we shall not distinguish them in the sequel. ♣

3.2.3 Basic closure and algorithmic results

We now generalize the convolution operator \otimes to take n trees ($n \in \mathbb{Z}_{\geq 1}$). Given k -ary trees $T_1 = (D_1, \tau_1), \dots, T_n = (D_n, \tau_n)$ over the labeling alphabet Σ , we define $T_1 \otimes \dots \otimes T_n$ to be the k -ary tree $T = (D, \tau)$ over the labeling alphabet Σ_{\perp}^n , where $\Sigma_{\perp} = \Sigma \cup \{\perp\}$ and $\perp \notin \Sigma$, such that

- $D = \bigcup_{i=1}^n D_i$, and
- for each $u \in D$, it is the case that $\tau(u) = (a_1, \dots, a_n)$, where

$$a_i = \begin{cases} \tau_i(u) & \text{if } u \in D_i, \\ \perp & \text{otherwise.} \end{cases}$$

Observe that when $n = 2$ this definition coincides with the 2-ary convolution operator for $\text{TREE}_k(\Sigma)$ that we defined earlier. An n -ary relation R over $\text{TREE}_k(\Sigma)$ is said to be *tree-regular* (or simply *regular*) if the language

$$\{T_1 \otimes \dots \otimes T_n : (T_1, \dots, T_n) \in R\}$$

is tree-regular. As before, we do not distinguish a relation and its language representation.

Definition 3.2.2 A $\text{TREE}_k(\Sigma)$ -automatic structure over the vocabulary σ is a structure $\mathfrak{S} = \langle S, \{R_a\}_{a \in \sigma} \rangle$, where S is a tree-regular language over $\text{TREE}_k(\Sigma)$ and R_a an $\text{AR}(a)$ -regular relation on S . A σ -structure is said to be *tree-automatic* if it is $\text{TREE}_k(\Sigma)$ -automatic for some integer $k > 0$ and labeling alphabet Σ .

A *presentation* η of a tree-automatic structure $\mathfrak{S}_{\eta} = \langle S, \{R_a\}_{a \in \sigma} \rangle$ is a simply tuple $\langle \mathcal{A}_S, \{\mathcal{A}_a\}_{a \in \sigma} \rangle$, where \mathcal{A}_S is an NTA over $\text{TREE}_k(\Sigma)$ with $\mathcal{L}(\mathcal{A}_S) = S$ and \mathcal{A}_a is an NTA over $\text{TREE}_k(\Sigma_{\perp}^n)$ (where $n = \text{AR}(a)$) such that $\mathcal{L}(\mathcal{A}_a) = R_a$. A σ -structure is said to be *tree-automatically presentable* if it is isomorphic to a tree-automatic structure over σ . Remark 3.2.1 for our definition of tree-automatic systems also holds for our definition of tree-automatic structures.

Example 3.2.4 It is easy to see that every Σ^* -automatic structure is a $\text{TREE}_1(\Sigma)$ -automatic structure. We now give two tree-automatically presentable structures which are *not* word-automatically presentable. The first is the structure (\mathbb{N}, \times) over natural numbers with multiplication (i.e. Skolem arithmetic). Each positive integer can be uniquely decomposed into a product of primes, say, $p_1^{a_1} \dots p_n^{a_n}$, where p_i is the i th

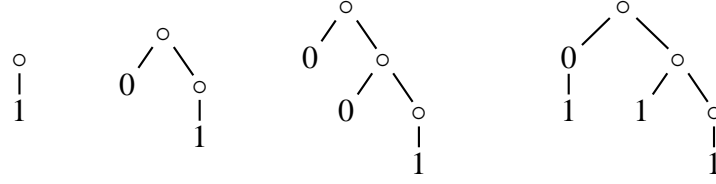


Figure 3.2: Tree representation of the numbers 2, 3, 5, and 60.

prime and $a_n > 0$. We may then represent each positive integer as a binary tree over the labeling alphabet $\{0, 1, \circ\}$, whose i th branch corresponds to the reverse binary representation of a_i . For example, the number 2, 3, 5 and 60 can be represented as the trees in Figure 3.2. The ternary relation \times can then be recognized by an NTA which separately runs the NWA for adding numbers on each branch (it is easy to treat the number 0 as a separate case). See [BGR10] for more details. Another example is the structure $\langle \text{TREE}_k(\Sigma), \preceq, \approx_{\text{dom}} \rangle$, with the tree extension relation \preceq and the equal tree domain relation \approx_{dom} . Furthermore, it is still tree-automatic when we extend this structure with the binary relations succ_i^a ($1 \leq i \leq k$ and $a \in \Sigma$), which extend *each* leaf of a tree by its i th child labeled a . See [BLN07] for more details. For more examples, see the recent survey [BGR10]. ♣

Just as for the case of word-automatic structures, the images of a tree-automatic structure under a first-order query is also effectively tree-regular. The proof of this result is identical to the word case and so is omitted.

Proposition 3.2.1 *Given a presentation η of a tree-automatic structure*

$$\mathfrak{S}_\eta = \langle S, \{R_a\}_{a \in \sigma} \rangle$$

over the vocabulary σ and a first-order query $\mathfrak{v}(\bar{x}) \leftarrow \varphi(\bar{y})$ over σ , the relation $\llbracket \mathfrak{v} \rrbracket_{\mathfrak{S}_\eta}$ is effectively tree-regular.

Just as in the case of word-automatic structures, a better complexity can be obtained when restricting to conjunctive queries.

Proposition 3.2.2 *Let*

$$\mathfrak{v}(x_1, \dots, x_m) \leftarrow \exists y_1, \dots, y_n \varphi$$

be a conjunctive query over σ , where

$$\varphi = R_{a_1}(\bar{z}_1) \wedge \dots \wedge R_{a_h}(\bar{z}_h)$$

is a conjunction of atomic formulas, where $\bar{z}_i \subseteq \bar{x} \cup \bar{y}$ for each $i = 1, \dots, h$. Given a presentation $\eta = \langle \mathcal{A}_S, \{\mathcal{A}_a\}_{a \in \sigma} \rangle$ of the $\text{TREE}_k(\Sigma)$ -automatic structure \mathfrak{S}_η , an NTA \mathcal{A}_v accepting $\llbracket v \rrbracket_{\mathfrak{S}_\eta}$ of size $O(\prod_{i=1}^h \|\mathcal{A}_i\|)$ can be computed in time $O(|\Sigma|^{m+n} \times \prod_{i=1}^h \|\mathcal{A}_i\|)$.

Example 3.2.5 Just as in the case of word-automatic systems, Proposition 3.2.2 can be used to show that reachability relations and strict reachability relations for tree-automatic systems are polynomially interdefinable. See Example 3.1.7. ♣

3.2.4 Negative results

Finally, since tree-automatic transition systems generalize word-automatic transition systems, the negative results from word-automatic transition systems carry over to tree-automatic transition systems. The Σ_1^0 and Σ_1^1 upper bounds for, respectively, reachability and recurrent reachability also easily carry over to the automatic case.

3.3 Other generic frameworks

In this section, we shall briefly discuss several other generic frameworks that have been considered in the literature and compare them with word/tree automatic transition systems. In particular, we will mention length-preserving word-automatic systems, rational transition systems, Presburger-definable transition systems, and ω -automatic transition systems.

3.3.1 Length-preserving word-automatic transition systems

A relation $R \subseteq \Sigma^* \times \Sigma^*$ is said to be *length-preserving* if $(v, w) \in R$ implies $|v| = |w|$. A word-automatic system $\mathfrak{S} = \langle S, \{\rightarrow_a\}_{a \in \text{ACT}} \rangle$ is said to be *length-preserving* if each relation \rightarrow_a is length-preserving. The class of length-preserving word-automatic systems are commonly considered in the literature of *regular model checking* (e.g. see [AJNS04, Bou01, BJNT00, BHV04, JN00, Nil05]), which aims to develop semi-algorithms for dealing with various verification problems over generic frameworks where the domains are represented as words or trees over some alphabet². Length-preserving automatic systems are known to be suitable for modeling *parameterized*

²Despite this, nowadays the term “regular model checking” seems to almost exclusively mean model checking over length-preserving word-automatic systems

systems, which are simply distributed protocols with a finite, but unbounded, number of processes (e.g. the thesis [Nil05] gives many examples).

Most verification problems for length-preserving word-automatic systems are always defined slightly differently in such a way that words of *every* length will have to be considered simultaneously. For example, to check safety for parameterized systems, it is necessary to be able to compute $post^*(\mathcal{L}(\mathcal{A}))$ or $pre^*(\mathcal{L}(\mathcal{A}))$ for an arbitrary NWA \mathcal{A} . Note that, unlike in the case of general word-automatic systems, computing $post^*(s_0)$ and $pre^*(s_0)$, for any given word $s_0 \in \Sigma^*$, is decidable since there are at most $2^{O(|s_0|)}$ reachable configurations from s_0 . In fact, this simple observation generalizes to most verification problems (e.g. temporal logic model checking) when considered over length-preserving automatic transition systems. On the other hand, the set $post^*(\mathcal{L}(\mathcal{A}))$ and $pre^*(\mathcal{L}(\mathcal{A}))$ need not be regular nor computable in general since it can be used to solve the halting problems for Turing machines (e.g. see [AJNS04, BJNT00, Nil05]).

So, how general is length-preserving automatic systems compared to the class of all word-automatic systems? Which verification problems for the general class of all word-automatic systems can be reduced to the length-preserving case? We have seen an answer to the first question: length-preserving automatic systems indeed are a general class of infinite systems, although there are only finitely many reachable configurations from any given configuration. Let us now briefly answer the second question. A verification problem for the class of all word-automatic systems that involves *only finite paths* can in some sense be reduced to a variant of the problem over length-preserving automatic systems. This includes checking reachability (i.e. safety). The reduction is done by treating the padding symbol \perp as a letter in the domain of the system and composing each resulting transition relation with the language $\left[\begin{smallmatrix} \perp \\ \perp \end{smallmatrix} \right]^*$. That way, checking whether a configuration v can reach another configuration w can be reduced to checking whether $w \perp^* \subseteq post^*(v \perp^*)$ in the resulting length-preserving automatic system. In contrast, such a reduction cannot be done for liveness problems including recurrent reachability or for temporal logic model checking. To explain this, consider as an example the problem of checking whether, for two given NWAs \mathcal{A} and \mathcal{A}' over the alphabet Σ and a length preserving Σ^* -automatic system \mathfrak{G} , there exists a configuration v (or for each configuration v) in the language $\mathcal{L}(\mathcal{A})$ for which it is the case that $v \in Rec(\mathcal{L}(\mathcal{A}'))$ is satisfied in \mathfrak{G} . Since there are only finitely many reachable configurations from any given configuration v , the infinite path witnessing $v \in Rec(\mathcal{L}(\mathcal{A}'))$ in \mathfrak{G} must eventually loop at some configuration $v' \in \mathcal{L}(\mathcal{A}')$. In fact, it

is easy to see that this problem is r.e. or co-r.e., which is in contrast to the highly undecidable problem of recurrent reachability for the class of all word-automatic systems (see Proposition 3.1.6). This gives a theoretical justification that length-preserving automatic transition systems are *not* suitable for dealing with liveness and temporal logic (e.g. LTL) model checking since most interesting classes of infinite-state systems (e.g. pushdown systems) could easily generate a simple infinite path.

3.3.2 Presburger transition systems

Presburger transition systems are transition systems whose domains and transition relations are definable in Presburger arithmetic. More precisely, a *Presburger transition system* is a transition system $\mathfrak{S} = \langle S, \{\rightarrow_a\}_{a \in \text{ACT}} \rangle$, where for some first-order formulas $\varphi(x_1, \dots, x_k)$ and $\psi_a(x_1, \dots, x_{2k})$ over $\langle \mathbb{N}, + \rangle$, for each $a \in \text{ACT}$, it is the case that

- $S = \{(i_1, \dots, i_k) \in \mathbb{N}^k : \langle \mathbb{N}, + \rangle \models \varphi(i_1, \dots, i_k)\}$, and
- for all tuples $(i_1, \dots, i_k), (j_1, \dots, j_k) \in \mathbb{N}^k$, we have $(i_1, \dots, i_k) \rightarrow_a (j_1, \dots, j_k)$ iff $\langle \mathbb{N}, + \rangle \models \psi_a(i_1, \dots, i_k, j_1, \dots, j_k)$.

A transition system is said to be *Presburger-presentable* if it is isomorphic to a Presburger transition system. Every presburger transition system is word-automatic [BG04, BHMV94] since the structure $\langle \mathbb{N}, + \rangle$ is word-automatic (as we saw in Example 3.1.6) and first-order queries are regularity preserving over automatic structures (Proposition 3.1.1). On the other hand, some word-automatic systems are not even Presburger-presentable. For example, it can be seen that the word-automatic transition system $\langle \{0, 1\}^*, \text{succ}_0, \text{succ}_1, \preceq \rangle$ is not Presburger-presentable since the first-order theory of $\langle \{0, 1\}^*, \text{succ}_0, \text{succ}_1, \preceq \rangle$ has a nonelementary lower bound [CH90, Sto74], while the first-order theory of every fixed Presburger-presentable system can be solved in 3-fold exponential time by any standard algorithm for checking satisfactions in Presburger arithmetic (e.g. see [Koz06]).

The class of Presburger transition systems (and restrictions thereof) has been considered as generic frameworks for model checking in the literature and many powerful semi-algorithms for computing reachable sets and reachability relations using representations of semilinear sets have been fully implemented (e.g. see [Boi99, YKB09, YKBB05, LAS, ABS01, BFLP08, FL02]). Although the class of Presburger transition systems is strictly smaller than the class of word-automatic systems, it still subsumes

many interesting classes of systems such as Minsky machines and Petri nets, many subclasses of which we shall encounter in the sequel. One of the most appealing aspects of the class of Presburger transition systems is that it contains many natural subclasses whose reachability relations are themselves Presburger-definable. This includes reversal-bounded counter systems [ISD⁺02, Iba78] and their extensions with one free counter and/or discrete clocks [ISD⁺02, Iba78, DIB⁺00]; and many subclasses of Petri nets [LS04, Esp97b, LS05a].

One reason to consider the full class of word-automatic transition systems instead of the subclass of Presburger transition systems is that there are natural models of computation whose reachability relations can be captured within word-automatic framework, but not within Presburger framework. Two such models include pushdown systems and prefix-recognizable systems. The framework of Presburger transition systems is not even powerful enough to capture the transition systems that are generated by prefix-recognizable systems; the system $\langle \{0, 1\}^*, \text{succ}_0, \text{succ}_1, \preceq \rangle$ is one such example. It turns out that the framework of Presburger transition systems is powerful enough to capture pushdown systems, although not their reachability relations.

Proposition 3.3.1 *Pushdown systems are Presburger transition systems*

Proof Sketch. Each configuration $(q_i, w) \in Q \times \Gamma^*$ of a pushdown system \mathcal{P} with states $Q = \{q_0, \dots, q_{n-1}\}$ and stack alphabet $\Gamma = \{0, 1\}$ can be interpreted (in a bijective way) as a tuple $(i, \mathbf{bin}(1w)) \in \mathbb{N} \times \mathbb{N}$. Note that the stack content is interpreted as $\mathbf{bin}(1w)$ instead of $\mathbf{bin}(w)$ so that $w = 000$ and $w = 0$ are not interpreted as the same numbers. The transition relations \rightarrow_a can also be easily defined as a formula $\varphi(x_1, y_1, x_2, y_2)$ in Presburger arithmetic. Obviously the changes in the finite state unit of \mathcal{P} can be handled easily in Presburger arithmetic. To deal with the changes in the stack content, first observe that testing whether w has a suffix of the form $u \in \{0, 1\}^*$ expressed using a sequence of divisibility tests by 2 (at most $|u|$ times), which is expressible in Presburger arithmetic. For example, to check whether w has a suffix of the form 01 can be done by testing that $2 \nmid y_1$ and $2 \mid \lfloor y_1/2 \rfloor$. The changes in the stack content can also be deal with using a sequence of divisions and multiplications by 2 (and perhaps additionally additions/subtractions by a 1). \square

3.3.3 Rational transition systems

Rational transition systems are transition systems whose domain is a regular set of words over some alphabet (like word-automatic systems) and whose transition rela-

tions are “rational”, which is a more general notion than regular relations. In order to define the notion of rational systems, we need to first recall the standard notion of finite-state input/output transducers (a.k.a. rational transducers). A *rational transducer* \mathcal{R} over the input alphabet Σ is an NWA over the alphabet $\Sigma_\epsilon \times \Sigma_\epsilon$, where $\Sigma_\epsilon = \Sigma \cup \{\epsilon\}$. The relation *realized* by \mathcal{R} consists of precisely all tuples $(v, w) \in \Sigma^* \times \Sigma^*$ where, for some $\begin{bmatrix} a_1 \\ b_1 \end{bmatrix} \dots \begin{bmatrix} a_n \\ b_n \end{bmatrix} \in \mathcal{L}(\mathcal{R}) \subseteq (\Sigma_\epsilon \times \Sigma_\epsilon)^*$, it is the case that $v = a_1 \dots a_n$ and $w = b_1 \dots b_n$. Note that in this case we do not necessarily have $|v| = |w|$ since some of the letters a_i ’s and b_j ’s might be ϵ . A simple example of a rational relation is the relation $\{(a^n, a^{2n}) : n \in \mathbb{N}\}$ over the alphabet $\Sigma = \{a\}$, which can be easily proved to be not a regular relation by an application of pumping lemma for regular languages. We refer the reader to the textbook [Ber79] for a more thorough treatment of rational transducers and their basic properties. A *rational transition system* is a transition system $\mathfrak{S} = \langle S, \{\rightarrow_a\}_{a \in \text{ACT}} \rangle$, where for some NWA \mathcal{A} and rational transducers $\{\mathcal{R}_a\}_{a \in \text{ACT}}$, it is the case that $S = \mathcal{L}(\mathcal{A})$ and \rightarrow_a is realized by \mathcal{R}_a . Rational transducers are also studied in the context of natural language processing (e.g. [JM00]).

The class of rational transition systems is more general than the class of word-automatic transition systems, but is not known to be comparable to the class of tree-automatic transition systems. On the other hand, unlike automatic systems, many simple problems are already undecidable for rational transition systems. For example, it is undecidable to check whether a rational transition system has a self-loop [Mor00], which is trivially decidable for word/tree automatic systems since the property can be easily expressed in first-order logic. Other such problems include checking whether a transition relation \rightarrow_a in the system is symmetric, reflexive, or transitive [Joh86], all of which are easily expressible in first-order logic. Despite this, model checking HM-logic enriched with inverse modality over rational transition systems and regular atomic propositions is decidable [BG09] since the images and preimages of regular languages under rational relations are effectively regular. A partial technique for computing the transitive closure of rational relations has also been proposed by Dams et al. in [DLS02].

In summary, although rational transition systems is a natural class of transition systems and is more general than the class of word-automatic transition systems, it remains to be seen whether it can be used to model natural classes of infinite-state transition systems with decidable model checking that cannot already be modeled as word-automatic systems. We also leave it as an open question if the positive results in this thesis can be extended in some way to the class of rational transition systems.

3.3.4 ω -word automatic transition systems

ω -word automatic transition systems can be defined in the same way as word-automatic transition systems, but using NBWAs instead of NWAs. They can be easily seen as a natural generalization of word-automatic transition systems (cf. [Bar07, BGR10, Blu99, BG04]). They are also known to satisfy most properties which are satisfied by word-automatic transition systems (e.g. closure under boolean combinations and automata projections, and decidability of first-order logic). Similar notions can also be defined for ω -word automatic structures. The class of ω -word automatic structures is quite expressive. For example, they include real numbers with addition $\langle \mathbb{R}, + \rangle$ even with an extra test of whether x is an integer [Blu99, BG04] and other interesting predicates. It follows that any infinite-state transition system which can be defined in the first-order theory of reals (possibly with extra tests for integers) are also ω -automatic. There are several interesting classes of infinite-state transition systems that are known to be definable in the first-order theory of the reals including real-timed systems (cf. [CJ99]). In fact, Comon and Jurski showed that even the reachability relation is definable in the first-order theory of the reals. For these reasons, it is natural to adopt ω -word automatic transition systems as a generic framework and consider whether the results in this thesis for word/tree automatic systems can be proven for ω -word automatic systems. We leave this as future work.

We shall also mention some partial techniques that have been developed for restrictions ω -word automatic systems. Legay *et al.* [BLW03, BLW09] have developed semi-algorithms for computing reachability relations and LTL model checking for ω -word automatic systems when the given automata are weak-deterministic. These results are orthogonal to the result in this thesis.

Chapter 4

Algorithmic metatheorems for recurrent reachability

Recall that the *problem of recurrent reachability over word (resp. tree) automatic systems* is defined as follows: given a presentation η of a word (resp. tree) automatic system $\mathfrak{S}_\eta = \langle S, \{\rightarrow_a\}_{a \in \text{ACT}} \rangle$, an initial configuration $s_0 \in S$, and a NWA (resp. NTA) \mathcal{A} (called the “target automaton”), decide whether $v_0 \in \text{Rec}(\mathcal{L}(\mathcal{A}))[\rightarrow^+]$. That is, we wish to decide whether there exists an infinite path from v_0 in \mathfrak{S}_η which visits $\mathcal{L}(\mathcal{A})$ infinitely often. The *global version* of this problem is simply to compute an NWA (resp. NTA) representing the set $\text{Rec}(\mathcal{L}(\mathcal{A}))[\rightarrow^+]$, if this set turns out to be regular. These problems are tightly connected with LTL model checking, as we shall see in the next chapter.

As we mentioned in Proposition 3.1.6, the problem of recurrent reachability is Σ_1^1 -complete (i.e. highly undecidable) for automatic systems. In this chapter, we shall show that it becomes decidable *if we are given an NWA (resp. NTA) representing the reachability relation of the automatic system as part of the input*. In fact, stronger results are shown in this chapter. Firstly, it turns out that in this case the set $\text{Rec}(\mathcal{L}(\mathcal{A}))[\rightarrow^+]$ is guaranteed to be effectively regular. Secondly, a succinct representation of the witnessing infinite path as nondeterministic Büchi word/tree automata can also be computed. As an immediate corollary, when we restrict to any subclass \mathcal{C} of word/tree automatic systems for which there exists an algorithm for computing the reachability relations, we immediately obtain decidability for recurrent reachability and its global version over \mathcal{C} . In this sense, our result is an algorithmic metatheorem for decidable recurrent reachability. The time complexity of our algorithm is polynomial in the size of the automaton \mathcal{R} which represents the reachability relation \rightarrow^+ of

the word/tree automatic system \mathfrak{S}_η and the size of the target automaton \mathcal{A} , on top of the time taken to produce \mathcal{R} from the input presentation η . We shall present our algorithmic metatheorem for word automatic systems in Section 4.1, and for tree-automatic systems in Section 4.2. In these sections, we shall also give direct applications of our algorithmic metatheorems for *uniformly* deriving a polynomial-time complexity for recurrent reachability and its global version for pushdown systems and (regular) ground tree rewrite systems, and an exponential-time complexity for prefix-recognizable systems. These turn out to be also optimal for the respective classes of systems. These results are already known in the literature (e.g. see [EKS03, KPV02, Löd03, Löd06]). In Chapter 6, we will use these algorithmic metatheorems for deriving new decidability results for checking recurrent reachability, e.g., over reversal-bounded counter systems extended with discrete clocks, PA-processes, and order-2 collapsible pushdown systems.

In Section 4.3, we give an extension of our algorithmic metatheorem for recurrent reachability to handle *generalized Büchi condition*: given several automata $\mathcal{A}_1, \dots, \mathcal{A}_n$ (instead of one, as for the original problem) decide whether there exists an infinite path visiting *each* of the sets $\mathcal{L}(\mathcal{A}_1), \dots, \mathcal{L}(\mathcal{A}_n)$ infinitely often. In fact, this extension is a comparatively simple corollary of our algorithmic metatheorem for recurrent reachability over word/tree automatic systems. In this case, the complexity of our algorithm becomes exponential in n and polynomial in the size of other input parameters, which we show to be optimal. Using this result, we derive an exponential-time algorithm for recurrent reachability with generalized Büchi condition for pushdown systems, for which we give a PSPACE lower bound, and prefix-recognizable systems and regular ground tree-rewrite systems, for which we give a matching EXP lower bound. Incidentally, this answers an open question by Löding [Löd06] concerning the decidability of recurrent reachability with generalized Büchi conditions¹.

In practice, it has been observed (cf. [AJNS04, AJRS06, BFLP08, BLW03, BLW09, DLS02, Nil05]) that partial techniques for computing the reachability relations over many generic frameworks have not been as successful in practice as partial techniques for deriving reachability sets, although recent work by Cook et al. (cf. [CPR06b, CPR06a]) sheds a light that they could be made practical. In Section 4.4, we study what we can deduce when we are given an automaton \mathcal{R} which represents an over/under approximation of the reachability relation \rightarrow^+ , i.e., $\rightarrow^+ \subseteq \mathcal{L}(\mathcal{R})$ or $\mathcal{L}(\mathcal{R}) \subseteq \rightarrow^+$. Such

¹Löding [Löd06] remarked that his technique for solving recurrent reachability for RGTRSs cannot easily handle generalized Büchi condition, which he left as an open problem whether it is still decidable

techniques could prove valuable when combined with partial techniques for computing upper/under approximations of the reachability relations of word/tree automatic systems.

In the next chapter, we shall apply our results from this chapter for deriving algorithmic metatheorems for decidable LTL model checking with complex fairness constraints, as well as model checking extensions of $\text{FO}_{\text{REG}}(\text{Reach})$ with recurrent reachability operators. Part of the result in this chapter has been published in [TL08].

4.1 The word-automatic case

In this section, we prove an algorithmic metatheorem for decidable recurrent reachability over Σ^* -automatic systems, which we will apply for deriving an optimal complexity of checking recurrent reachability over PDSs and prefix-recognizable systems. Analyzing the proof of our main theorem, we shall also deduce that with an extra polynomial-time overhead we may compute four “small” words over Σ which represent periodic infinite paths witnessing positive instances of the problem.

4.1.1 Main theorem

The setting of our algorithmic metatheorems is simple. Let us start with a class \mathcal{C} of presentations of word-automatic systems. What conditions are sufficient for ensuring decidable recurrent reachability over \mathcal{C} ? We shall now give one such sufficient condition.

Definition 4.1.1 *A class \mathcal{C} of presentations of automatic systems is said to be closed under transitive closure if, for each automatic transition system $\mathfrak{S}_\eta = \langle S, \{\rightarrow_a\}_{a \in \text{ACT}} \rangle$ presented by some $\eta \in \mathcal{C}$ over Σ , the transitive closure \rightarrow^+ of $(\bigcup_{a \in \text{ACT}} \rightarrow_a) \subseteq \Sigma^* \times \Sigma^*$ is regular. Furthermore, the class \mathcal{C} is effectively closed under transitive closure if there exists an algorithm $\mathcal{M}_{\mathcal{C}}$ that computes an NWA \mathcal{R}^+ recognizing this transitive closure relation for each input presentation $\eta \in \mathcal{C}$. We say that $\mathcal{M}_{\mathcal{C}}$ is an effective transitive closure witness (ETC-witness) of \mathcal{C} .*

Notice that above we may alternatively define that the non-strict reachability relations \rightarrow^* are effectively regular. However, these two definitions are equivalent since these two relations are polynomially interdefinable (see Example 3.1.7). In the sequel, we shall tacitly assume that $\mathcal{M}_{\mathcal{C}}$ runs in at least linear time since such an algorithm should

read the entire input. The output of the algorithm \mathcal{M} on input η is written $\mathcal{M}(\eta)$ with size $\|\mathcal{M}(\eta)\|$, which obviously satisfies $\|\mathcal{M}(\eta)\| \leq \text{TIME}_{\mathcal{M}}(\|\eta\|)$. We now state our algorithmic metatheorem for decidable recurrent reachability.

Theorem 4.1.1 *Suppose \mathcal{C} is class of automatic systems closed under transitive closure. Then, given a presentation $\eta \in \mathcal{C}$ over Σ of an automatic system $\mathfrak{S}_\eta = \langle S, \{\rightarrow_a\}_{a \in \text{ACT}} \rangle$ and an NWA \mathcal{A} over Σ , the set $\text{Rec}(\mathcal{L}(\mathcal{A}))$ is regular.*

Moreover, if \mathcal{C} is effectively closed under transitive closure with an ETC-witness \mathcal{M} , then an NWA recognizing $\text{Rec}(\mathcal{L}(\mathcal{A}))$ of size $O((\|\mathcal{M}(\eta)\| + \|\eta\|) \times \|\mathcal{M}(\eta)\| \times \|\mathcal{A}\|)$ is computable in time $\text{TIME}_{\mathcal{M}}(\|\eta\|) + O(|\Sigma|^2 \times \|\mathcal{M}(\eta)\|^3 \times \|\mathcal{A}\|^2)$.

Observe that once an NWA \mathcal{A}' recognizing $\text{Rec}(\mathcal{L}(\mathcal{A}))$ has been computed, we can check whether $v \in \text{Rec}(\mathcal{L}(\mathcal{A}))$ for a given word $v \in \Sigma^*$ in time $O(|\Sigma| \times |v| \times \|\mathcal{A}'\|)$ by a standard membership algorithm for NWAs. In other words, assuming effective closure under transitive closure for the class \mathcal{C} of presentations of automatic systems, recurrent reachability is decidable in polynomial time assuming that the reachability relation is given as part of the input.

4.1.2 Proof of the main theorem

We now give a proof of Theorem 4.1.1. Firstly, our assumption of closure under transitive closure gives an NWA \mathcal{R} over the alphabet $\Sigma_\perp \times \Sigma_\perp$ recognizing the transitive closure \rightarrow^+ of $(\bigcup_{a \in \text{ACT}} \rightarrow_a)$. By definition, we have $v \in \text{Rec}(\mathcal{L}(\mathcal{A}))$ iff there exists a sequence $\{v_i\}_{i \in \mathbb{N}}$ of words in Σ^* with $v_0 = v$ such that $v_{i-1} \otimes v_i \in \mathcal{L}(\mathcal{R})$ and $v_i \in \mathcal{L}(\mathcal{A})$ for all $i > 0$. We now divide the set $\text{Rec}(\mathcal{L}(\mathcal{A}))$ into two sets $\text{Rec}_\circ(\mathcal{L}(\mathcal{A}))$ and $\text{Rec}_\rightarrow(\mathcal{L}(\mathcal{A}))$, where $\text{Rec}_\circ(\mathcal{L}(\mathcal{A}))$ contains words with witnessing sequence $\{v_i\}_{i \in \mathbb{N}}$ that satisfies $v_j = v_k$ for some $k > j \geq 0$, and $\text{Rec}_\rightarrow(\mathcal{L}(\mathcal{A}))$ contains words with a witnessing sequence $\{v_i\}_{i \in \mathbb{N}}$ that satisfies $v_j \neq v_k$ for all distinct $j, k \in \mathbb{N}$. Clearly, it is the case that

$$\text{Rec}(\mathcal{L}(\mathcal{A})) = \text{Rec}_\circ(\mathcal{L}(\mathcal{A})) \cup \text{Rec}_\rightarrow(\mathcal{L}(\mathcal{A})).$$

Hence, we may separately construct NWAs for $\text{Rec}_\circ(\mathcal{L}(\mathcal{A}))$ and $\text{Rec}_\rightarrow(\mathcal{L}(\mathcal{A}))$, from which we can easily compute their union. Let us start with the easy case of constructing an NWA \mathcal{A}_\circ recognizing $\text{Rec}_\circ(\mathcal{L}(\mathcal{A}))$.

Lemma 4.1.2 *An NWA \mathcal{A}_\circ of size at most $\|\mathcal{A}\| \times \|\mathcal{R}\| \times (\|\mathcal{R}\| + \|\eta\|)$ recognizing $\text{Rec}_\circ(\mathcal{L}(\mathcal{A}))$ can be constructed in time $O(|\Sigma|^2 \times \|\mathcal{A}\| \times \|\mathcal{R}\| \times (\|\mathcal{R}\| + \|\eta\|))$.*

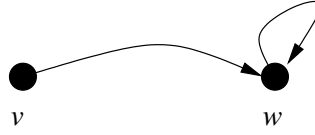


Figure 4.1: A witnessing sequence for $v \in \text{Rec}_{\cup}(\mathcal{L}(\mathcal{A}))$ of lasso shape.

Proof. Observe that, for each word $v \in \Sigma^*$, $v \in \text{Rec}_{\cup}(\mathcal{L}(\mathcal{A}))$ iff there exists a word $w \in \Sigma^*$ such that $v \rightarrow^* w$, $w \rightarrow^+ w$, and $w \in \mathcal{L}(\mathcal{A})$. The pair (v, w) is a witnessing sequence of lasso shape; see Figure 4.1. To this end, we simply apply Proposition 3.1.2 on the formula

$$\phi(x) := \exists y (x \rightarrow^* y \wedge y \rightarrow^+ y \wedge y \in \mathcal{L}(\mathcal{A})).$$

Observe that an NWA for \rightarrow^* can be obtained by taking a union of the NWA \mathcal{R} for \rightarrow^+ and the NWA (of size $\|\eta\|$) for $(\bigcup_{a \in \text{ACT}} \rightarrow_a)$. Therefore, we obtain an upper bound of $\|\mathcal{A}\| \times \|\mathcal{R}\| \times (\|\mathcal{R}\| + \|\eta\|)$ for the size of \mathcal{A}_{\cup} , and an upper bound of $O(|\Sigma|^2 \times \|\mathcal{A}\| \times \|\mathcal{R}\| \times (\|\mathcal{R}\| + \|\eta\|))$ for the amount of time needed to compute \mathcal{A}_{\cup} . \square

Thus, it remains to construct an NWA $\mathcal{A}_{\rightarrow}$ recognizing $\text{Rec}_{\rightarrow}(\mathcal{L}(\mathcal{A}))$.

Lemma 4.1.3 *An NWA $\mathcal{A}_{\rightarrow}$ of size $O(\|\mathcal{A}\| \times \|\mathcal{R}\|^2)$ recognizing $\text{Rec}_{\rightarrow}(\mathcal{L}(\mathcal{A}))$ can be constructed in time $O(|\Sigma|^2 \times \|\mathcal{A}\|^2 \times \|\mathcal{R}\|^3)$.*

The proof of this lemma is substantially more involved than the proof of the previous lemma. Therefore, we first give the proof idea. Firstly, by applying pigeonhole principles on word lengths, we can show that it suffices to consider witnessing infinite sequences $\{v_i\}_{i \in \mathbb{N}}$ such that there exist two sequences $\{\alpha_i\}_{i \in \mathbb{N}}$ and $\{\beta_i\}_{i \in \mathbb{N}}$ of words over Σ such that:

1. $|\alpha_i| > 0$ for all $i > 0$,
2. $|\alpha_i| = |\beta_i|$ for all $i \in \mathbb{N}$, and
3. $v_i = \beta_0 \dots \beta_{i-1} \alpha_i$ for all $i \in \mathbb{N}$.

See Figure 4.2 for an illustration of witnessing infinite sequences of this special form. Such pairs of sequences $\{\alpha_i\}_{i \in \mathbb{N}}$ and $\{\beta_i\}_{i \in \mathbb{N}}$ can then be represented as a pair (α, β) of ω -words over $\Sigma \cup \{\#\}$, where $\#$ is a new symbol not in Σ and

$$\begin{aligned} \alpha &:= \alpha_0 \# \alpha_1 \# \dots, \\ \beta &:= \beta_0 \# \beta_1 \# \dots \end{aligned}$$

$$\begin{aligned}
v_0 &= \alpha_0 \\
v_1 &= \beta_0 \alpha_1 \\
v_2 &= \beta_0 \beta_1 \alpha_2 \\
v_3 &= \beta_0 \beta_1 \beta_2 \alpha_3 \\
&\vdots
\end{aligned}$$

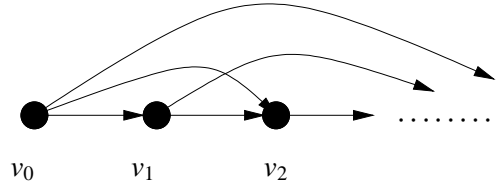
Figure 4.2: Witnessing infinite sequences $\{v_i\}_{i \in \mathbb{N}}$ of special form. The lengths of the words in the sequence are strictly increasing and that, for all $i \in \mathbb{N}$, $|\alpha_i| = |\beta_i|$.

The strategy then is to construct an NBWA \mathcal{B} that accepts ω -words $\alpha \otimes \beta$ corresponding to witnessing sequences of this special form. Once \mathcal{B} is constructed, it will be easy to obtain $\mathcal{A}_{\rightarrow}$. If we assume that the NWAs \mathcal{A} and \mathcal{R} are deterministic, the construction of \mathcal{B} is then rather immediate. We will, however, *refrain from determinizing* the NWAs \mathcal{A} and \mathcal{R} since this will cause an exponential blow-up in the size of the automaton \mathcal{B} . Instead, by further applying pigeonhole principles on the runs of \mathcal{A} and infinite Ramsey theorem on the runs of \mathcal{R} , we will prove sufficiency of infinite sequences of the above special form that satisfy further technical restrictions (see below) as witnesses. An NBWA \mathcal{B} that recognizes such sequences can then be constructed in polynomial time.

We shall now elaborate the details of the proof of Lemma 4.1.3. Let $\mathcal{A} = (\Sigma_{\perp} \times \Sigma_{\perp}, Q, \delta, q_0, F)$ and $\mathcal{R} = (\Sigma, Q', \delta', q'_0, F')$. The following lemma asserts that we may restrict ourselves to witnessing infinite sequences of a special form.

Lemma 4.1.4 *For every word $v \in \Sigma^*$, it is the case that $v \in \text{Rec}_{\rightarrow}(\mathcal{L}(\mathcal{A}))$ iff there exist two infinite sequences $\{\alpha_i\}_{i \in \mathbb{N}}$ and $\{\beta_i\}_{i \in \mathbb{N}}$ of words over Σ such that*

- (1) $\alpha_0 = v$ and $|\alpha_i| > 0$ for all $i > 0$,
- (2) $|\alpha_i| = |\beta_i|$ for all $i \in \mathbb{N}$,
- (3) *there exists an infinite run π of \mathcal{A} on $\beta_0 \beta_1 \dots$ such that, for all $i \in \mathbb{N}$, the NWA \mathcal{A} accepts α_{i+1} from q , where $q = \pi(|\beta_0 \dots \beta_i|)$,*
- (4) *there exists an infinite run π' of \mathcal{R} on $(\beta_0 \times \beta_0)(\beta_1 \otimes \beta_1) \dots$ such that, for all $i \in \mathbb{N}$, \mathcal{R} accepts $\alpha_i \otimes \beta_i \alpha_{i+1}$ from q' where $q' = \pi'(|\beta_0 \dots \beta_{i-1}|)$.*

Figure 4.3: An illustration of ω -chains.

One direction of the lemma is easy: if (1)–(4) hold, then from the infinite sequences $\{\alpha_i\}_{i \in \mathbb{N}}$ and $\{\beta_i\}_{i \in \mathbb{N}}$ we can form a new sequence $\{v_i\}_{i \in \mathbb{N}}$ with $v_i := \beta_0 \dots \beta_{i-1} \alpha_i$. Condition (3) ensures that $v_i \in \mathcal{L}(\mathcal{A})$ for all $i > 0$, and condition (4) implies that $v_i \rightarrow^+ v_{i+1}$ for all $i \in \mathbb{N}$. This implies that $v \in \text{Rec}_{\rightarrow}(\mathcal{L}(\mathcal{A}))$ and thus proving sufficiency in Lemma 4.1.4. To prove the converse, we will prove a more general lemma concerning ω -chains, which are simply the transitive closures of any one-directional infinite path (see Figure 4.3). More precisely, let $R \subseteq \Sigma^* \times \Sigma^*$ be a (not necessarily transitive) binary relation and $U \subseteq \Sigma^*$ a language. A (transitive) U -coloured ω -chain in R from a word $v \in \Sigma^*$ is an infinite sequence $\{v_i\}_{i \in \mathbb{N}}$ of *distinct* words in Σ^* such that the following three properties are satisfied:

- $v_0 = v$,
- for each integer $i > 0$, it is the case that $v_i \in U$, and
- for each pair of integers $j \geq i \geq 0$, we have $(v_i, v_j) \in R$.

Figure 4.3 gives an illustration of ω -chains. We write $\text{CHAIN}(U, R)$ to denote the set of words $v \in \Sigma^*$ from which there exists a U -coloured ω -chain in R . Observe that $\text{CHAIN}(\mathcal{L}(\mathcal{A}), \rightarrow^+)$ coincides with $\text{Rec}_{\rightarrow}(\mathcal{L}(\mathcal{A}))$. The following proposition is a Ramsey-type result for ω -chains in word-automatic systems.

Proposition 4.1.5 *Suppose that \mathcal{N} is an NWA over Σ , and \mathcal{T} an NWA over $\Sigma_{\perp} \times \Sigma_{\perp}$ recognizing a regular relation $R \subseteq \Sigma^* \times \Sigma^*$. Then, for every word*

$$v \in \text{CHAIN}(\mathcal{L}(\mathcal{N}), \mathcal{L}(\mathcal{T})),$$

there exists a word $v'v''$ such that

1. $|v'| = |v|$ and $|v''| > 0$
2. $v \otimes v'v'' \in \mathcal{L}(\mathcal{T})$,

3. there exists an accepting run ρ of \mathcal{N} on $v'v''$, and a run ρ' on \mathcal{T} on $v' \otimes v'$ such that $v'' \in \text{CHAIN}(\mathcal{L}(\mathcal{N}^q), \mathcal{L}(\mathcal{T}^{q'}))$, where $q := \rho(|v|)$, $q' := \rho'(|v|)$, and \mathcal{N}^q (resp. $\mathcal{T}^{q'}$) is the NWA \mathcal{N} but with q (resp. q') as the initial state.

Proof. Suppose that $v \in \text{CHAIN}(\mathcal{L}(\mathcal{N}), \mathcal{L}(\mathcal{T}))$. Then, there exists a sequence $\sigma = \{v_i\}_{i \in \mathbb{N}}$ of distinct words over Σ such that $v_0 = v$, and it is the case that, for all $i > 0$, the word v_i is in $\mathcal{L}(\mathcal{N})$ with accepting run η_i , and for all distinct pair of indices $j > i \geq 0$, we have $v_i \otimes v_j \in \mathcal{L}(\mathcal{T})$. As there are only finitely many different words of length $|v|$ but infinitely many words in σ , we may assume that $|v_i| > |v|$ for all $i \geq 1$; for, otherwise, we may simply omit these words from σ . Now every word v_i , where $i > 0$, can be written as $v_i = u_i w_i$ for some words u_i, w_i such that $|u_i| = |v|$ and $|w_i| > 0$. As there are only finitely many different words of length $|v|$ and finitely many different runs of \mathcal{N} of length $|v|$, by pigeonhole principle there must exist $k > 0$ such that $\eta_j[0, |v|] = \eta_k[0, |v|]$ (and so $u_j = u_k$ by the definition of NWA runs) for infinitely many $j > 0$. Let $v' := u_k$ and $\rho := \eta_k[0, |v|]$. Therefore, we may discard all words v_i in σ with $i \geq 1$ such that η is not a prefix of η_i . By renaming indices, call the resulting sequence $\sigma = \{v_i\}_{i \in \mathbb{N}}$ and, for all $i \geq 1$, denote by η_i the accepting run of \mathcal{N} on v_i that has ρ as a prefix. Notice that σ is still a witness for $v \in \text{CHAIN}(\mathcal{L}(\mathcal{N}), \mathcal{L}(\mathcal{T}))$. So, for $k > j \geq 0$, let $\theta_{j,k}$ denote the accepting run of \mathcal{T} on $v_j \otimes v_k$. Let \mathcal{X} denote the *finite* set of all runs of \mathcal{T} on $v' \otimes v'$. Notice that it is not necessarily the case that $|\mathcal{X}| = 1$ since \mathcal{T} is nondeterministic. Therefore, consider the edge-labeled undirected graph $\mathfrak{G} = \langle V, \{E_u\}_{u \in \mathcal{X}} \rangle$ such that $V = \mathbb{Z}_{>0}$ and

$$E_u = \{\{j, k\} : 0 < j < k \text{ and } u \text{ is a prefix of } \theta_{j,k}\}.$$

Notice that $\{E_u\}_{u \in \mathcal{X}}$ is a partition of $\{\{j, k\} : j \neq k, k > 0\}$, and so \mathfrak{G} is a complete graph. By infinite Ramsey theorem, G has a monochromatic complete infinite subgraph $H = \langle V', E_u \rangle$ for some $u \in \mathcal{X}$. Set $\rho' := u$. Notice that if the elements of V' are $i_1 < i_2 < \dots$, then the run θ_{i_j, i_k} (for all $k > j > 0$) has u as a prefix. Therefore, we can discard all words v_i ($i > 0$) in σ such that $i \notin V'$ and by renaming indices call the resulting sequence $\sigma = \{v_i\}_{i \in \mathbb{N}}$. We now set v'' to be the unique word w such that $v_1 = v'w$. It is easy to see that (1) and (2) are satisfied. Furthermore, it is easy to check that $v'' \in \text{CHAIN}(\mathcal{L}(\mathcal{N}^q), \mathcal{L}(\mathcal{T}^{q'}))$ with a witnessing sequence $\{w_i\}_{i > 0}$, where w_i is the unique word such that $v_i = v'w_i$ for all $i > 0$. \square

Now it is not difficult to complete the proof of Lemma 4.1.4. Given $v \in \text{Rec}_{\rightarrow}(\mathcal{L}(\mathcal{A})) = \text{CHAIN}(\mathcal{L}(\mathcal{A}), \mathcal{L}(\mathcal{T}))$, we will inductively construct the desired sequences $\{\alpha_i\}_{i \in \mathbb{N}}$

and $\{\beta_i\}_{i \in \mathbb{N}}$, along with the run π of the NWA \mathcal{A} and the run π' of the NWA \mathcal{R} , by using Proposition 4.1.5 at every induction step. The gist of the proof is that from the word $v'v''$ given by Proposition 4.1.5 at induction step k , we will set $\beta_k = v'$ and $\alpha_{k+1} = v''$, and extend the partial runs π and π' in Lemma 4.1.4. Notice that we have $v'' \in \text{CHAIN}(\mathcal{L}(\mathcal{N}^q), \mathcal{L}(\mathcal{T}^{q'}))$, which sets up the next induction step. See the appendix for a detailed argument. This completes the proof of Lemma 4.1.4.

It is now easy to construct an NBWA \mathcal{B} that recognizes ω -words of the form $\alpha \otimes \beta$ satisfying

$$\alpha := \alpha_0 \# \alpha_1 \# \dots,$$

$$\beta := \beta_0 \# \beta_1 \# \dots$$

for some $\{\alpha_i\}_{i \in \mathbb{N}}$ and $\{\beta_i\}_{i \in \mathbb{N}}$ satisfying the conditions in Lemma 4.1.4. The automaton \mathcal{B} will attempt to simultaneously guess the runs π and π' , while at the same time checking that the runs satisfy the conditions (3) and (4) in Lemma 4.1.4. To this end, \mathcal{B} will run a copy of \mathcal{A} and \mathcal{R} , while simultaneously also running several other copies of \mathcal{A} and \mathcal{R} to check that the runs π and π' guessed so far satisfy the conditions (3) and (4) along the way. The automaton \mathcal{B} consists of three components depicted as Box 1, Box 2, and Box 3 in Figure 4.4. The first box is used for reading the prefix of the input before the first occurrence of $\begin{bmatrix} \# \\ \# \end{bmatrix}$, while the other boxes are used for reading the remaining suffix. Boxes 2-3 are essentially identical, i.e., they have the same sets of states and essentially the same transition functions. When \mathcal{B} arrives in Box 2, it will read a single letter in $\Sigma \times \Sigma$ and goes to Box 3 so as to make sure that $|\alpha_i| > 0$ for each $i > 0$. When \mathcal{B} is in Box 3, it will go to Box 2 upon reading the letter $\begin{bmatrix} \# \\ \# \end{bmatrix}$. We will set all states in Box 2 as the final states so as to make sure that infinitely many $\begin{bmatrix} \# \\ \# \end{bmatrix}$ is seen, i.e., the sequences $\{\alpha_i\}_i$ and $\{\beta_i\}_i$ are both infinite, and each words α_i and β_i are finite.

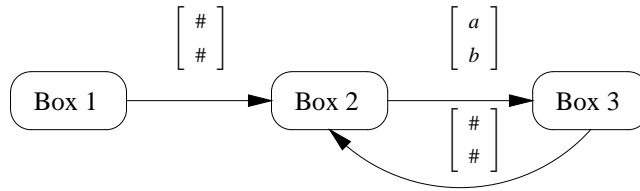


Figure 4.4: A bird's eye view of the Büchi automaton \mathcal{B}

More precisely, the NBWA $\mathcal{B} = \left(\Sigma^2 \cup \left\{ \begin{bmatrix} \# \\ \# \end{bmatrix} \right\}, S, \Delta, s_0, S_F \right)$ is defined as follows. Define

$$P := (Q \times Q' \times Q') \cup (Q \times Q' \times Q \times Q' \times Q' \times \{2, 3\}).$$

Intuitively, $Q \times Q' \times Q'$ will be the states in Box 1 and $Q \times Q' \times Q \times Q' \times Q' \times \{i\}$ will be the states in Box i . The initial state is defined to be $s_0 := (q_0, q'_0, q'_0)$. The first and the last components in each state are meant for guessing the infinite runs π and π' . The second component of each state in Box 1 is used for guessing a prefix of the accepting run of \mathcal{R} on $\alpha \otimes \beta_0 \alpha_1$. The automaton \mathcal{B} will finish this guessing when it reaches Box 3 upon the completion of parsing $\alpha_1 \otimes \beta_1$. When \mathcal{B} is in Box 2 or 3 reading $\alpha_i \otimes \beta_i$, where $i > 0$, the third and fourth components of the states are used for checking that $\beta_0 \dots \beta_{i-1} \alpha_i \otimes \beta_0 \dots \beta_i \alpha_{i+1} \in \mathcal{L}(\mathcal{R})$, which will be completed in the next iteration. We now formally define the transition function. Let

$$\Delta \left((q, q', q''), \begin{bmatrix} a \\ b \end{bmatrix} \right) := \begin{cases} \delta(q, b) \times \delta' \left(q', \begin{bmatrix} a \\ b \end{bmatrix} \right) \times \delta' \left(q'', \begin{bmatrix} b \\ b \end{bmatrix} \right) & , \text{ if } a, b \neq \# \\ (q, q'', q, q', q'', 2) & , \text{ if } a = b = \# \\ \emptyset & , \text{ otherwise.} \end{cases}$$

and, when \mathcal{B} is in a state in $Q \times Q' \times Q \times Q' \times Q' \times \{i\}$, where $i = 2, 3$, and $a, b \neq \#$ we define

$$\begin{aligned} \Delta \left((q_1, q_2, q'_1, q'_2, q''_2, i), \begin{bmatrix} a \\ b \end{bmatrix} \right) &:= \delta(q_1, b) \times \delta' \left(q_2, \begin{bmatrix} a \\ b \end{bmatrix} \right) \times \delta(q'_1, a) \times \\ &\delta' \left(q'_2, \begin{bmatrix} \perp \\ a \end{bmatrix} \right) \times \delta' \left(q''_2, \begin{bmatrix} b \\ b \end{bmatrix} \right) \times \{3\}. \end{aligned}$$

If $q'_1 \in F$ and $q'_2 \in F'$, then we set

$$\Delta \left((q_1, q_2, q'_1, q'_2, q''_2, 3), \begin{bmatrix} \# \\ \# \end{bmatrix} \right) = (q_1, q'_2, q_1, q_2, q'_2, 2).$$

Finally, the set of final states are $S_F := Q \times Q' \times Q \times Q' \times Q' \times \{2\}$. Correctness of this construction is immediate. Furthermore, it is easy to see that the construction takes time $O(|\Sigma|^2 \times \|\mathcal{A}\|^2 \times \|\mathcal{R}\|^3)$.

Now, from \mathcal{B} we can easily compute the NWA $\mathcal{A}_{\rightarrow} = (Q^1, \Sigma, \delta^1, q_0^1, F^1)$ that recognizes $Rec_{\rightarrow}(\mathcal{L}(\mathcal{A}))$. The automaton $\mathcal{A}_{\rightarrow}$ accepts the set of finite words α_0 such that the word $\alpha_0 \# \alpha_1 \# \dots \otimes \beta_0 \# \beta_1 \# \dots$ is accepted by \mathcal{B} for some $\{\alpha_i\}_{i \geq 0}$ and $\{\beta_i\}_{i \in \mathbb{N}}$. Therefore, we will set the new set of states Q^1 to be $Q \times Q' \times Q'$, i.e., the first component of \mathcal{B} in Figure 4.4. We then apply projection operation on the transition function Δ of \mathcal{B}

to obtain δ^1 . More precisely, if $a \in \Sigma$, we set

$$\delta^1((q_1, q_2, q'_2), a) = \bigcup_{b \in \Sigma} \Delta \left((q_1, q_2, q'_2), \begin{bmatrix} a \\ b \end{bmatrix} \right).$$

Finally, the new set F^1 of final states will be those states in Q^1 from which \mathcal{B} can accept some ω -words of the form $\begin{bmatrix} \# \\ \# \end{bmatrix} w$ for some ω -word w . For this, a simple modification of the standard linear-time algorithm for testing nonemptiness for NBWA can be applied (cf. Proposition 2.2.6), which still takes linear time. Finally, it is easy to check that the size of the resulting NWA is $O(\|\mathcal{A}\| \times \|\mathcal{R}\|^2)$, and the total time taken to compute it is $O(|\Sigma|^2 \times \|\mathcal{R}\|^3 \times \|\mathcal{A}\|^2)$ on top of the time taken to compute \mathcal{R} . This completes the proof of Lemma 4.1.3 and hence the proof of Theorem 4.1.1.

Remark 4.1.1 As we mentioned earlier, the proof of Lemma 4.1.3 can be greatly simplified by determinizing the NWA \mathcal{R} . By doing this, we avoid the use of Ramsey theorem, but at the expense of an exponential blow-up, i.e., the algorithm no longer runs in polynomial time. Such a proof technique (without Ramsey theorem) was used in [KRS05] for proving a König's lemma for automatic partial orders. In fact, our proof above directly improves the complexity of the results from [KRS05]. ■

4.1.3 A small witness for recurrent reachability

The following corollary of the proof of Theorem 4.1.1 is rather immediate.

Corollary 4.1.6 *Suppose that \mathcal{C} is a class of automatic systems effectively closed under transitive closure with an ETC-witness \mathcal{M} . Then, given a presentation $\eta \in \mathcal{C}$ over Σ of an automatic system $\mathfrak{S}_\eta = \langle S, \{\rightarrow_a\}_{a \in \text{ACT}} \rangle$, a word $v_0 \in S$, and an NWA \mathcal{A} over Σ , we can effectively decide whether $v_0 \in \text{Rec}(\mathcal{L}(\mathcal{A}))$ and, if so, produce a witness of the form:*

1. $(v_0, w) \in \Sigma^* \times \Sigma^*$ such that $v_0 \rightarrow^* w$, $w \rightarrow^+ w$, and $w \in \mathcal{L}(\mathcal{A})$, or
2. $(v_0, w_0, v_1, w_1) \in (\Sigma^*)^4$ such that whenever $s_0 := v_0$ and $s_i := w_0 w_1^{i-1} v_1$ (for each integer $i \geq 1$), it is the case that:
 - $s_i \rightarrow^+ s_j$ for each pair of integers $j > i \geq 0$, and
 - $s_i \in \mathcal{L}(\mathcal{A})$ for each integer $i > 0$.

Furthermore, the total running time of the algorithm is $\text{TIME}_{\mathcal{M}}(\|\eta\|) + O(|\Sigma|^2 \times \|\mathcal{M}(\eta)\|^3 \times \|\mathcal{A}\|^2 \times |v_0|)$.

We shall now sketch the proof of this corollary. Whenever $v_0 \in \text{Rec}(\mathcal{L}(\mathcal{A}))$, we either have $v_0 \in \text{Rec}_{\circlearrowleft}(\mathcal{L}(\mathcal{A}))$ or $v_0 \in \text{Rec}_{\rightarrow}(\mathcal{L}(\mathcal{A}))$. For the former case, as in the proof of Lemma 4.1.2, we may apply the construction from the Proposition 3.1.1 on the formula

$$\Psi(y) := v_0 \rightarrow^* y \wedge y \rightarrow^+ y \wedge y \in \mathcal{L}(\mathcal{A})$$

and obtain an NWA $\mathcal{A}'_{\circlearrowleft}$ of size $O(|v_0| \times \|\mathcal{A}\| \times \|\mathcal{R}\| \times (\|\mathcal{R}\| + \|\eta\|))$. Therefore, we may easily compute a witnessing word $w \in \mathcal{L}(\mathcal{A}'_{\circlearrowleft})$ such that $|w| \leq \|\mathcal{A}'_{\circlearrowleft}\|$. For the case $v_0 \in \text{Rec}_{\rightarrow}(\mathcal{L}(\mathcal{A}))$, we work from the NBWA \mathcal{B} that was obtained during the computation of $\mathcal{A}_{\rightarrow}$. From \mathcal{B} , we may compute a new NBWA \mathcal{B}' that recognizes ω -words $\alpha \otimes \beta \in \mathcal{L}(\mathcal{B})$ of the form

$$\alpha := v_0 \# \alpha_1 \# \dots$$

$$\beta := \beta_0 \# \beta_1 \# \dots$$

for some sequences $\{\alpha_i\}_{i>0}$ and $\{\beta_i\}_{i \in \mathbb{N}}$ of words in Σ^* . The NBWA \mathcal{B}' can be computed by taking a product of Box 1 in Figure 4.4 with a simple NWA that recognizes only the word v_0 . To obtain the words $w_0, v_1, w_1 \in \Sigma^*$, we simply apply the standard nonemptiness algorithm for NBWA on \mathcal{B}' . In fact, the maximum length of these words is bounded by the size of the NBWA \mathcal{B}' . The complexity of the algorithm in Corollary 2 is immediate from the complexity of the algorithm from Theorem 4.1.1.

4.1.4 Two appetizer examples

We now give two immediate applications of Theorem 4.1.1 for deriving an optimal complexity (up to a polynomial) of checking recurrent reachability of pushdown systems and prefix-recognizable systems. More concrete examples will be given in later chapters.

Pushdown systems

Recall that pushdown systems can be thought of as word-automatic systems (see Example 3.1.2). Caucal [Cau90] proved that the reachability relations of pushdown systems are rational, for which a rational transducer is computable in polynomial time. Later in [Cau92] Caucal noted that the reachability relations are in fact also regular, for which NWAs can be computed within the same time complexity.

Proposition 4.1.7 ([Cau90, Cau92]) *Given a PDS \mathcal{P} , the reachability relation \rightarrow^* of \mathcal{P} is regular. Furthermore, an NWA for \rightarrow^* can be computed in time polynomial in $\|\mathcal{P}\|$.*

Notice that this proposition immediately implies the regularity of the strict reachability relation \rightarrow^+ , for which an NWA can be computed in polynomial time (see Example 3.1.7). Combining this with Theorem 4.1.1, the following theorem is immediate.

Theorem 4.1.8 *Recurrent reachability for PDSs is decidable in polynomial time. Furthermore, the set of configurations $\text{Rec}(\mathcal{L}(\mathcal{A}))$ which satisfies the recurrent reachability properties is also regular for which an NWA is computable in polynomial time.*

This theorem is not new. In fact, it can be inferred from either from the result of Löding [Löd03, Löd06] concerning recurrent reachability of ground tree rewrite systems or the result of Esparza, Kucera and Schwon [EKS03] concerning LTL model checking over PDSs with “regular valuations”.

For the sake of completeness, we shall now sketch a proof of Proposition 4.1.7. We start with the following well-known result, which can be proven using the standard “saturation” construction (e.g. see [BEM97, EHS00]).

Proposition 4.1.9 *Given a PDS \mathcal{P} and an NWA \mathcal{A} , one can compute in polynomial time two automata \mathcal{A}_{pre^*} and \mathcal{A}_{post^*} recognizing $pre^*(\mathcal{L}(\mathcal{A}))$ and $post^*(\mathcal{L}(\mathcal{A}))$, respectively.*

In fact, the algorithm given in [EHS00] computes these automata in cubic time, and the sizes of \mathcal{A}_{pre^*} and \mathcal{A}_{post^*} are at most quadratic in $\|\mathcal{A}\|$. Now, given a PDS $\mathcal{P} = (\text{ACT}, \Gamma, Q, \delta)$, we write $\text{Dom}(\mathcal{P})$ for the set of configurations $qu \in Q\Gamma^*$ such that $((q, a, u), (q', u')) \in \delta$ for some $a \in \text{ACT}$, $q' \in Q$ and $u' \in \Gamma^*$. To construct an NWA \mathcal{R} recognizing the reachability relation of \mathcal{P} , we shall need the following easy lemma.

Lemma 4.1.10 *Given a pushdown system $\mathcal{P} = (\text{ACT}, \Gamma, Q, \delta)$ and two configurations $q_1u_1, q_2u_2 \in Q\Gamma^*$, then $q_1u_1 \rightarrow^* q_2u_2$ iff there exists a configuration q_3u_3 of \mathcal{P} , which satisfies either $q_3u_3 \in \text{Dom}(\mathcal{P})$ or $u_3 = \varepsilon$, and words $x, v_1, v_2 \in \Gamma^*$ such that $u_1 = xv_1$, $u_2 = xv_2$, and $q_1v_1 \rightarrow^* q_3u_3 \rightarrow^* q_2v_2$.*

This lemma can be easily proved by induction on the length of the path witnessing $q_1u_1 \rightarrow^* q_2u_2$. Now constructing the NWA \mathcal{R} for the reachability relation \rightarrow^* of \mathcal{P} is simple. First, we use Proposition 4.1.9 to compute the NWAs $\mathcal{A}_{pre^*}^C$ and $\mathcal{A}_{post^*}^C$ that recognize, respectively, $pre^*(C)$ and $post^*(C)$ for every configuration $C \in \text{Dom}(\mathcal{P}) \cup Q$.

Then, on input $q_1u_1 \otimes q_2u_2$, the NWA \mathcal{R} first remembers (q_1, q_2) in its finite memory, and *guesses* a configuration $C \in \text{Dom}(\mathcal{P}) \cup Q$ and a position at which the initial common prefix x in Lemma 4.1.10 ends. The automaton \mathcal{R} then simultaneously runs the automata $\mathcal{A}_{pre^*}^C$ and $\mathcal{A}_{post^*}^C$ to verify that the top part v_1 and the bottom part v_2 of the remaining input word (preceding the padding symbol \perp) satisfy $q_1v_1 \in \mathcal{L}(\mathcal{A}_{pre^*}^C)$ and $q_2u_2 \in \mathcal{L}(\mathcal{A}_{post^*}^C)$. By Proposition 4.1.9, NWAs for $pre^*(C)$ and $post^*(C)$ can be computed in polynomial time for each configuration $C \in \text{Dom}(\mathcal{P}) \cup Q$. Hence, we can also compute the NWA \mathcal{R} in polynomial time.

Prefix-recognizable systems

We now use Theorem 4.1.1 to infer the decidability of recurrent reachability for prefix-recognizable systems with optimal complexity. Recall that prefix-recognizable systems can be thought of as word-automatic systems (see Example 3.1.3). It turns out that the reachability relations for prefix-recognizable systems are also regular, for which NWAs can be computed in exponential time.

Proposition 4.1.11 *Given a prefix-recognizable system $\|\mathcal{P}\|$, the reachability relation \rightarrow^* of \mathcal{P} is regular. Furthermore, an NWA for \rightarrow^* can be computed in time exponential in $\|\mathcal{P}\|$.*

As for PDSs, this proposition also implies the regularity of the strict reachability relations of prefix-recognizable systems, for which NWAs can be computed within the same time complexity. To prove the above proposition, we first use the well-known fact (e.g. see [Cac02, Löd03]) that given a prefix-recognizable system $\mathcal{P} = (\text{ACT}, \Gamma, Q, \delta)$ we could construct another prefix-recognizable system $\mathcal{P}' = (\text{ACT}, \Gamma', Q, \delta')$ such that each rule $((q, a, \mathcal{A}), (q', \mathcal{A}'), \mathcal{A}'')$ satisfies $\mathcal{L}(\mathcal{A}'') = \Gamma^*$. In other words, the *prefix constraints* in the original prefix-recognizable systems have been removed. This fact can be proven via the standard technique for prefix-rewrite systems (cf. [Cac02, EKS03, Löd03]) of annotating the runs of *each* NWA representing a prefix constraint of \mathcal{P} in the new alphabet Γ' and each rule in δ' , which causes the size of Γ' and δ' to be exponential in the number of prefix constraints in \mathcal{P} (but polynomial in the rest of the parameters). One could now proceed in exactly the same way as in the proof of Proposition 4.1.7 of the previous example, and infer that the reachability relation for \mathcal{P}' is regular, for which an NWA can be computed in time polynomial in $\|\mathcal{P}\|$.

Combining Proposition 4.1.11 and Theorem 4.1.1, we obtain the desired result on recurrent reachability for prefix-recognizable systems.

Theorem 4.1.12 *Recurrent reachability for prefix-recognizable systems is decidable in exponential time. Furthermore, the set $\text{Rec}(\mathcal{L}(\mathcal{A}))$ which satisfies the recurrent reachability properties is also regular for which an NWA is computable in exponential time.*

This theorem was previously known, e.g., it can be easily derived from the result of Kupferman, Piterman, and Vardi [KPV02]. In addition, it turns out that the complexity given in the preceding theorem is optimal in the sense that the problem is EXP-hard, which can be easily deduced from the recent result of Göller [Göl08] on the EXP-completeness of the reachability problem for prefix-recognizable systems.

4.2 The tree-automatic case

In this section, we extend our algorithmic metatheorem for recurrent reachability over word-automatic systems to tree-automatic systems, which we apply for deriving an optimal complexity of checking recurrent reachability over regular ground tree rewrite systems. The proof in this section is substantially more technical than the proof from the previous section since we need to reason about many different branches of the trees at the same time. Nonetheless, the proof idea is essentially the same: we encode an infinite sequence of trees as an infinite tree which will be recognized by a nondeterministic top-down Büchi automaton. Therefore, the reader is advised to first read the construction from the previous section.

4.2.1 Main theorem

Let us first state the tree analogue of Definition 4.1.1.

Definition 4.2.1 *A class \mathcal{C} of presentations of tree-automatic systems is said to be closed under transitive closure if, for each tree-automatic system $\mathfrak{S}_\eta = \langle S, \{\rightarrow_a\}_{a \in \text{ACT}} \rangle$ presented by some $\eta \in \mathcal{C}$, the transitive closure \rightarrow^+ of $(\bigcup_{a \in \text{ACT}} \rightarrow_a)$ is tree-regular. Furthermore, the class \mathcal{C} is effectively closed under transitive closure if there exists an algorithm $\mathcal{M}_\mathcal{C}$ that computes an NTA \mathcal{R}^+ recognizing this transitive closure relation for each input presentation $\eta \in \mathcal{C}$. We say that $\mathcal{M}_\mathcal{C}$ is an effective transitive closure witness (ETC-witness) of \mathcal{C} .*

As before, we shall tacitly assume that $\mathcal{M}_\mathcal{C}$ runs in at least linear time since such an algorithm should read the entire input. The output of the algorithm \mathcal{M} on input η is

written $\mathcal{M}(\eta)$ with size $\|\mathcal{M}(\eta)\|$, which obviously satisfies $\|\mathcal{M}(\eta)\| \leq \text{TIME}_{\mathcal{M}}(\|\eta\|)$. We now state our algorithmic metatheorem for decidable recurrent reachability for tree-automatic systems.

Theorem 4.2.1 *Suppose that \mathcal{C} is a class of tree-automatic presentations that are closed under transitive closure. Then, given a presentation $\eta \in \mathcal{C}$ of a $\text{TREE}_k(\Sigma)$ -automatic system $\mathfrak{S}_\eta = \langle S, \{\rightarrow_a\}_{a \in \text{ACT}} \rangle$ and an NTA \mathcal{A} over $\text{TREE}_k(\Sigma)$, it is the case that the set $\text{Rec}(\mathcal{L}(\mathcal{A}))$ is tree-regular.*

Moreover, if \mathcal{C} is effectively closed under transitive closure with an ETC-witness \mathcal{M} , then an NTA recognizing $\text{Rec}(\mathcal{L}(\mathcal{A}))$ of size $O(\|\mathcal{A}\| \times \|\mathcal{M}(\eta)\|^2)$ is computable in time $\text{TIME}_{\mathcal{M}}(\|\eta\|) + O(|\Sigma|^2 \times \|\mathcal{R}\|^6 \times \|\mathcal{A}\|^4)$.

Observe that the time complexity obtained in this theorem, albeit still polynomial, is slightly worse than the time complexity obtained in the word case (see Theorem 4.1.1).

4.2.2 Preliminaries

We will first fix some definitions and notations that we will use in the proof of Theorem 4.2.1. A k -ary (linear) context tree over the labeling alphabet Σ with variables $\mathcal{X} = \{x_1, \dots, x_n\}$ is a tree $T = (D, \tau) \in \text{TREE}_k(\Sigma \cup \mathcal{X})$ such that for each $i = 1, \dots, n$, there is exactly one node $u_i \in D$ with $\tau(u_i) = x_i$; furthermore, u_i is a leaf. The leaves u_1, \dots, u_n are also called *context leaves*. To emphasize which variables are in T' , we will often write $T[x_1, \dots, x_n]$ for T . Observe that whenever $n = 0$ the context tree T is just a normal tree (a.k.a. *ground tree*). Given ground trees $t_1, \dots, t_n \in \text{TREE}_k(\Sigma)$, the tree $T[t_1, \dots, t_n]$ is the ground tree $T[t_1/u_1, \dots, t_n/u_n]$, obtained by replacing all context leaves u_1, \dots, u_n by the ground trees t_1, \dots, t_n , respectively. We also define $T \otimes T$ just as we defined the operator \otimes for ground trees, but we replace the label $\begin{bmatrix} x_i \\ x_i \end{bmatrix}$ by x_i . For a context tree $T' = (D', \tau')$ and a tree $T = (D, \tau)$, we write $T' \preceq T$ if $D' \subseteq D$ and $\tau'(u) = \tau(u)$ whenever $u \in D'$ and u is not a context leaf.

Given an NTA $\mathcal{A} = (\Sigma, Q, \delta, q_0, F)$ over $\text{TREE}_k(\Sigma)$, we now extend the notion of runs of \mathcal{A} to k -ary context trees $T = (D, \tau)$ over Σ with variables $\mathcal{X} = \{x_1, \dots, x_n\}$ and context leaves u_1, \dots, u_n . First, we define $\mathbf{virt}(T)$ to be the k -ary tree $T' = (D', \tau')$ over the alphabet $\Sigma' := \Sigma \cup \mathcal{X} \cup \{\$, \}$, where $\$ \notin \Sigma$, such that $D' = D \cup \{v_i : v \in D \setminus \{u_1, \dots, u_n\}, 1 \leq i \leq k\}$ and

$$\tau'(u) = \begin{cases} \tau(u) & \text{if } u \in D, \\ \$ & \text{otherwise.} \end{cases}$$

Observe that this definition is to a large extent similar to the special case of ground trees, except that we treat the context leaves u_1, \dots, u_n in the original context tree T as virtual leaves. A *run* of \mathcal{A} on T , then, is a mapping $\rho : D' \rightarrow Q'$ that can be defined in the same way as for ground trees by treating u_1, \dots, u_n as virtual leaves. We say that ρ is *potentially accepting* if $\rho(u) \in F$ for each leaf $u \in D' - \{u_1, \dots, u_n\}$. In other words, potentially accepting runs *might* become accepting after we replace the context leaves with some ground trees.

We shall also need the definition of *unranked trees* as a conceptual tool. An *unranked tree* over a potentially infinite labeling alphabet Σ is a tree over the labeling alphabet Σ and some direction alphabet $Y = \{0, \dots, k\}$, for some integer $k > 1$, with a potentially infinite tree domain. Notice that the direction alphabet of unranked trees are not *a priori* fixed, although they are finitely branching.

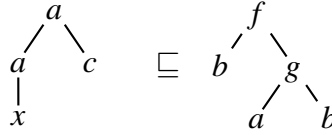
4.2.3 Proof of the main theorem

Let $Y = \{1, \dots, k\}$. Let \mathcal{R} be the NTA over $\text{TREE}_k(\Sigma_\perp) \times \text{TREE}_k(\Sigma_\perp)$ that recognizes the transitive closure \rightarrow^+ of $(\bigcup_{a \in \text{ACT}} \rightarrow_a)$. For the rest of the proof, we write $\mathcal{A} = (Q_1, \delta_1, q_0^1, F_1)$ and $\mathcal{R} = (Q_2, \delta_2, q_0^2, F_2)$. By definition, for every tree $T \in \text{TREE}_k(\Sigma)$, we have $T \in \text{Rec}(\mathcal{A})$ iff there exists an infinite sequence $\{T_i\}_{i \in \mathbb{N}}$ of trees in $\text{TREE}_k(\Sigma)$ such that $T_0 = T$, $T_{i-1} \rightarrow^+ T_i$, and $T_i \in \mathcal{L}(\mathcal{A})$ for all $i > 0$. As in the case of words, we shall prove that it is sufficient to consider only infinite sequences of trees of a special form that can be recognized by a Büchi infinite-tree automaton \mathcal{B} , after which constructing the desired NTA \mathcal{A}' for $\text{Rec}(\mathcal{A})$ will be easy. Unlike in the word case, we shall find it notationally simpler *not* to treat separately trees with looping and non-looping witnessing sequences.

Just as in the word case, we shall apply pigeonhole principles on the *structure* of the subtrees in the witnessing infinite sequence to obtain a witnessing infinite sequence in a special form. The main difference in the tree case is that the number of branches (as well as the length thereof) of the trees appearing in the witnessing sequence could all grow indefinitely. To this end, we shall need the following definition.

Definition 4.2.2 For any context tree $T'[x_1, \dots, x_n] = (D', \tau') \in \text{TREE}_k(\Sigma \cup \{x_1, \dots, x_n\})$ and a tree $T = (D, \tau) \in \text{TREE}_k(\Sigma)$, we write $T'[x_1, \dots, x_n] \sqsubseteq T$ (or just $T' \sqsubseteq T$) if, whenever u_1, \dots, u_n are the context leaves in T labeled by x_1, \dots, x_n , respectively, it is the case that

- for each $i = 1, \dots, n$, we have $u_i \notin D$,

Figure 4.5: An illustration of the relation \sqsubseteq .

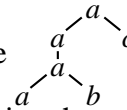
- $D' - \{u_1, \dots, u_n\} \subseteq D$, and
- $u_i = v_i r_i$ for some $r_i \in \Upsilon$ and $v_i \in D \cap D'$.

In other words, $T' \sqsubseteq T$ if all the nodes in T' are in T except for the context leaves. See Figure 4.5 for an example. Note that in the word case the relation \sqsubseteq simply reduces to comparing the length of two words. The following lemma gives a basic fact about the relation \sqsubseteq .

Lemma 4.2.2 *Given trees $T_1, T_2 \in \text{TREE}_k(\Sigma)$, there exists a context tree T with variables x_1, \dots, x_n (for some $n \in \mathbb{N}$) such that the following two conditions hold:*

- (1) $T[x_1, \dots, x_n] \sqsubseteq T_2$, and
- (2) for some trees $t_1, \dots, t_n \in \text{TREE}_k(\Sigma)$, it is the case that $T[t_1, \dots, t_n] = T_1$.

Furthermore, the context tree T is unique up to relabeling of the context leaves.

As an illustration of Lemma 4.2.2, we may take T_1 to be the tree  and T_2 to be the right tree in Figure 4.5. The unique context tree T satisfying the two prescribed conditions in Lemma 4.2.2 is the left tree in Figure 4.5. The proof of the lemma is easy, which we relegate into the appendix. We are now ready to state a normal form lemma (analogous to Lemma 4.1.4) for the infinite sequences witnessing $T \in \text{Rec}(\mathcal{L}(\mathcal{A}))$.

Notation. For the rest of the proof, we shall use the following notation. Given an NTA $\mathcal{N} = (\Sigma, Q, \delta, q_0, F)$ over $\text{TREE}_k(\Sigma)$ and a state $q \in Q$, we write \mathcal{N}^q for the NTA $(\Sigma, Q, \delta, q, F)$ obtained by replacing the initial state of \mathcal{N} with q . ■

Lemma 4.2.3 *For every tree $T \in \text{TREE}_k(\Sigma)$, it is the case that $T \in \text{Rec}(\mathcal{L}(\mathcal{A}))$ iff there exists an unranked tree $\mathfrak{T} = (D_{\mathfrak{T}}, \tau_{\mathfrak{T}})$ over the labeling alphabet*

$$\Gamma := \{(t, t'[x_1, \dots, x_r], q, q') : q \in Q_1, q' \in Q_2, r \in \mathbb{N}, t \in \text{TREE}_k(\Sigma), \\ t' \in \text{TREE}_k(\Sigma \cup \{x_1, \dots, x_r\}), \text{ and } t' \sqsubseteq t\},$$

and $\tau_{\mathfrak{T}}(u) = (\alpha_u, \beta_u[x_1, \dots, x_{r_u}], q_u, q'_u)$ for all $u \in D_{\mathfrak{T}}$, such that the following conditions hold:

1. $\tau_{\mathfrak{T}}(\varepsilon) = (T, \beta_{\varepsilon}[x_1, \dots, x_{r_{\varepsilon}}], q_0^1, q_0^2)$ for some context tree $\beta_{\varepsilon}[x_1, \dots, x_{r_{\varepsilon}}]$ and some $r_{\varepsilon} \in \mathbb{N}$ such that $\beta_{\varepsilon} \sqsubseteq T$,
2. for all $u \in D_{\mathfrak{T}}$ we have
 - (a) the number of children of u is the same as r_u ,
 - (b) $\alpha_u \otimes \beta_u[\alpha_{u1}, \dots, \alpha_{ur_u}] \in \mathcal{L}(\mathcal{R}^{q'_u})$,
 - (c) if v_1, \dots, v_{r_u} are the nodes of β_u labeled by x_1, \dots, x_{r_u} respectively, then there exist an accepting run ρ_u of \mathcal{A}^{q_u} on $\beta_u[\alpha_{u1}, \dots, \alpha_{ur_u}]$ and a potentially accepting run ρ'_u of $\mathcal{R}^{q'_u}$ on $\beta_u \otimes \beta_u$ such that, for each $i = 1, \dots, r_u$, it is the case that $q_{ui} = \rho_u(v_i)$ and $q'_{ui} = \rho'_u(v_i)$.

Intuitively, the unranked tree \mathfrak{T} in the above lemma encodes an infinite sequence of a special form that witnesses $T \in \text{Rec}(\mathcal{L}(\mathcal{A}))$. In case of word-automatic systems, the tree \mathfrak{T} reduces to a single branch which may grow indefinitely. However, in general the tree \mathfrak{T} could have more than one infinite branch, which correspond to the branches in the witnessing infinite sequence that grow indefinitely. It is of course possible that all of the branches in \mathfrak{T} are finite (and hence \mathfrak{T} is finite) in which case each leaf of \mathfrak{T} is labeled by some $(t \otimes t'[x_1, \dots, x_r], q, q') \in \Gamma$ with $r = 0$. The role of α 's and β 's in the node labels of \mathfrak{T} is very similar to the word case (see Figure 4.2). We shall now show sufficiency in Lemma 4.2.3. To this end, we shall construct a witnessing sequence $\{T_i\}_{i \geq 0}$ out of the tree \mathfrak{T} . We shall inductively define $\{T_i\}_{i \geq 0}$ together with a sequence $\{C_i\}_{i \geq 0}$ of context trees as follows. We set $T_0 := \alpha_{\varepsilon} = T$, $C_0 := x$, $T_1 := \beta_{\varepsilon}[\alpha_1, \dots, \alpha_{r_{\varepsilon}}]$, and $C_1 := \beta_{\varepsilon}[x_1^{\varepsilon}, \dots, x_{r_{\varepsilon}}^{\varepsilon}]$. Suppose that C_i , for some $i \geq 1$, has been defined to be the context tree $T'[x_1^{u_1}, \dots, x_{r_{u_1}}^{u_1}, \dots, x_1^{u_n}, \dots, x_{r_{u_n}}^{u_n}]$ for all nodes u_1, \dots, u_n in \mathfrak{T} of level $i-1$, where $n \in \mathbb{N}$ and $r_{u_1}, \dots, r_{u_n} \in \mathbb{N}$. We define C_{i+1} to be $T'[\sigma]$, where σ replaces $x_k^{u_j}$ by $\beta_{u_j k}[x_1^{u_{jk}}, \dots, x_{r_{u_{jk}}}^{u_{jk}}]$. Similarly, we define T_{i+1} to be $T'[\sigma]$, where σ replaces $x_k^{u_j}$ by $\alpha_{u_{jk}}$. See Figure 4.6 for an illustration. Notice that if C_i is ground, then $T_{i+1} = T_i$ and $C_{i+1} = C_i$. By induction, the sequence $\{T_i\}_{i \geq 0}$ together with $\{C_i\}_{i \geq 0}$ have been defined. It is not difficult to prove by induction that $T_i \in \mathcal{L}(\mathcal{A})$ and $T_{i-1} \otimes T_i \in \mathcal{L}(\mathcal{R})$ for all $i \in \mathbb{Z}_{\geq 1}$. Therefore, we conclude that $T \in \text{Rec}(\mathcal{L}(\mathcal{A}))$.

We shall now prove the converse of Lemma 4.2.3. To this end, we shall prove a tree analogue of Proposition 4.1.5. Recall from Section 2.5 that $\text{Rec}(\mathcal{L}(\mathcal{A}))[R]$ is defined even when the relation R is non-transitive.

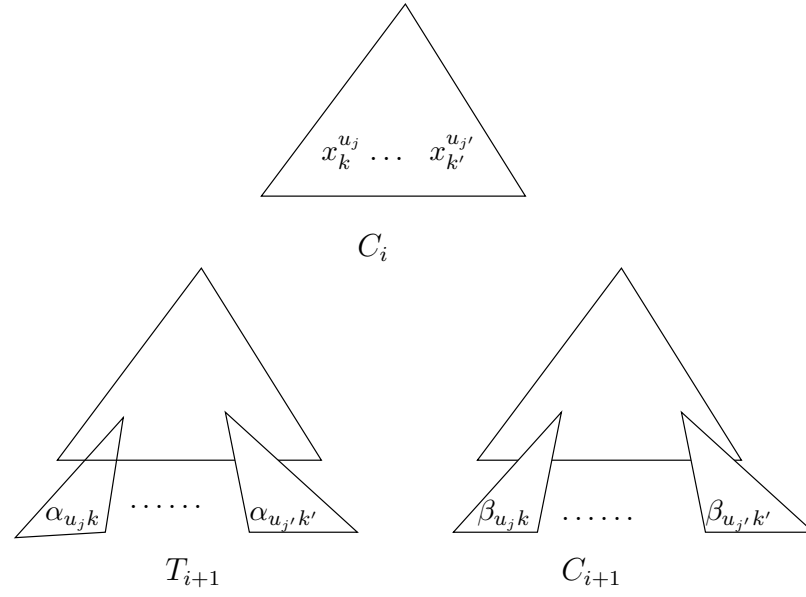


Figure 4.6: An illustration of how the trees C_{i+1} and T_{i+1} are obtained from C_i .

Proposition 4.2.4 *Suppose \mathcal{N} and \mathcal{T} are, respectively, an NTA over $\text{TREE}_k(\Sigma)$ and an NTA over $\text{TREE}_k(\Sigma_\perp \times \Sigma_\perp)$, where the relation $\mathcal{L}(\mathcal{T})$ is not necessarily transitive. For every tree $T = (D, \tau) \in \text{TREE}_k(\Sigma)$, if $T \in \text{Rec}(\mathcal{L}(\mathcal{N}))[\mathcal{L}(\mathcal{T})]$, then one of the following is true:*

- (1) *there exists a tree $T' = (D', \tau') \in \text{TREE}_k(\Sigma)$ such that $D' \subseteq D$, $T' \in \mathcal{L}(\mathcal{N})$, $(T, T') \in \mathcal{L}(\mathcal{T})$, and $(T', T') \in \mathcal{L}(\mathcal{T})$.*
- (2) *There exist a k -ary context tree $T'[x_1, \dots, x_n] = (D', \tau')$ over the labeling alphabet Σ and trees $t_1, \dots, t_n \in \text{TREE}_k(\Sigma)$ such that*
 - (a) $T' \sqsubseteq T$,
 - (b) $(T, T'[t_1, \dots, t_n]) \in \mathcal{L}(\mathcal{T})$,
 - (c) *there exist an accepting run $\rho = (D_\rho, \tau_\rho)$ of \mathcal{N} on $T'[t_1, \dots, t_n]$ and a potentially accepting run $\rho' = (D_{\rho'}, \tau_{\rho'})$ of \mathcal{T} on $T' \otimes T'$ such that, whenever $1 \leq i \leq n$, it is the case that $t_i \in \text{Rec}(\mathcal{L}(\mathcal{N}^{q_i}))[\mathcal{L}(\mathcal{T}^{q'_i})]$ where $q_i = \tau_\rho(u_i)$ and $q'_i = \tau_{\rho'}(u_i)$.*

Note that statement (1) yields an infinite witnessing sequence of lasso shape, as was shown in Figure 4.1 in the word case.

Proof. Suppose that $T = (D, \tau) \in \text{Rec}(\mathcal{L}(\mathcal{N}))[\mathcal{L}(\mathcal{T})]$, but statement (1) is false. Then, there exists an infinite sequence $\sigma = \{T_i\}_{i \in \mathbb{N}}$ of trees such that $T_0 = T$, $T_j \neq T_k$ for all distinct indices j, k (since statement (1) is false), and it is the case that, for all $i > 0$, $T_i \in \mathcal{L}(\mathcal{N})$ with accepting run $\eta_i = (D_{\eta_i}, \tau_{\eta_i})$, and for all distinct pair of indices $0 \leq i < i'$, $T_i \otimes T_{i'} \in \mathcal{L}(\mathcal{T})$. Now, for every tree T_i , where $i > 0$, Lemma 4.2.2 implies that there exists a unique context tree $C_i[x_1, \dots, x_{n_i}] = (D_i, \tau_i)$ with variables x_1, \dots, x_{n_i} for some $n_i \in \mathbb{N}$, such that $C_i \sqsubseteq T$, and $T_i = C_i[t_1^i, \dots, t_{n_i}^i]$ for some (ground) trees $t_1^i, \dots, t_{n_i}^i \in \text{TREE}_k(\Sigma)$. Let $H = \{C_i[x_1, \dots, x_{n_i}] : i > 0\}$. For infinitely many $i > 0$, it is the case that $n_i > 0$, i.e., there exists a node in T_i that is not in D ; for, otherwise, there are infinitely many indices i such that $D_i \subseteq D$ where $T_i = (D_i, \tau_i)$ and, since there are only finitely many different such trees, pigeonhole principle tells us that one of these trees must repeat in σ , which contradicts our assumption that statement (1) is false. On the other hand, it is easy to see that the number of nodes in any context tree C_i in H is bounded by $|Y| \times |D|$. Therefore, the set H is finite and so is the number of different potentially accepting runs of \mathcal{N} on context trees in H . So, if we define $\eta'_i := (\eta_i)_{|D_i}$, i.e., the part of the run tree η_i restricted to the domain D_i of C_i , then by pigeonhole principle there exists $k > 0$ such that $C_k[x_1, \dots, x_{n_k}] = C_j[x_1, \dots, x_{n_j}]$ and $\eta'_k = \eta'_j$ for infinitely many indices j s. Let $n := n_k$, $T'[x_1, \dots, x_n] := C_k[x_1, \dots, x_n]$, and $\eta' = \eta'_k$. We remove all elements T_i ($i > 0$) from σ such that $C_i \neq T'$ or $\eta'_i \neq \eta'$ and, by renaming indices, call the resulting sequence $\sigma = \{T_i\}_{i \in \mathbb{N}}$ where $T_0 = T$. The same is done for the sequence $\{\eta_i\}_{i \geq 1}$ of runs so that η_i is an accepting run \mathcal{N} on T_i ($i \geq 1$) such that $\eta' \preceq \eta_i$. Notice that σ is still a witness for $T \in \text{Rec}(\mathcal{L}(\mathcal{N}))[\mathcal{L}(\mathcal{T})]$. Now let $\theta_{j,k}$, where $0 \leq j < k$, be an accepting run of \mathcal{T} on $T_j \otimes T_k$. Let \mathcal{C} be the *finite* set of all potentially accepting runs of \mathcal{T} on $T' \otimes T'$. The set \mathcal{C} is nonempty as $T' \otimes T' \preceq T_j \otimes T_k$ and $T_j \otimes T_k \in \mathcal{L}(\mathcal{T})$. Consider the edge-labeled undirected graph $G = (V, \{E_\theta\}_{\theta \in \mathcal{C}})$ such that $V = \mathbb{Z}_{\geq 1}$ and

$$E_\theta := \{\{j, k\} : 0 < j < k \text{ and } \theta \preceq \theta_{j,k}\}.$$

Notice that $\{E_\theta\}_{\theta \in \mathcal{C}}$ is a partition of $\{\{j, k\} : j \neq k \in \mathbb{Z}_{\geq 1}\}$, and so G is a complete graph. By (infinite) Ramsey theorem, G has a monochromatic complete infinite subgraph $H = (V', E_{\rho'})$ for some $\rho' \in \mathcal{C}$. Notice that if V' contains precisely the elements $j_1 < j_2 < \dots$ then $\rho' \preceq \theta_{j_k, j_{k'}}$ for all $k' > k \geq 1$. We now remove all T_i ($i \geq 1$) from σ with $i \notin V'$ and, again, rename indices. Notice that σ is still a witness for $T \in \text{Rec}(\mathcal{L}(\mathcal{N}))[\mathcal{L}(\mathcal{T})]$. Recall that for each $i \geq 1$, we have $T_i = T'[t_1^i, \dots, t_n^i]$ for some ground trees t_1^i, \dots, t_n^i . Set $\rho := \eta_1$ and $t_k := t_1^k$ for each $k = 1, \dots, n$. Letting $\sigma_k = \{t_i^k\}_{i \geq 1}$ for each $k = 1, \dots, n$, it is easy now to check that $t_k \in \text{Rec}(\mathcal{L}(\mathcal{N}^{q_k}))[\mathcal{L}(\mathcal{T}^{q'_k})]$

with witnessing sequence σ_k , where $q_k = \tau_p(u_k)$ and $q'_k = \tau_{p'}(u_k)$ if u_k is the leaf node of T' labeled by x_k . So, condition (2c) holds. That (2b) holds is also immediate. As we already saw that $T' \sqsubseteq T$, our proof is complete. \square

In the same way we used Lemma 4.1.5 to complete the proof of necessity in Lemma 4.1.4, we can now finish off the proof of necessity in Lemma 4.2.3 by constructing the tree \mathfrak{T} inductively and adding nodes of height n at step $n \in \mathbb{N}$ by using Proposition 4.2.4. Therefore, the proof of Lemma 4.2.3 is complete.

Before we construct a Büchi tree automaton recognizing witnessing infinite paths of a special form, we will first show how any unranked tree \mathfrak{T} satisfying the conditions in Lemma 4.2.3 can be represented as ranked trees. For any tree $T = (D, \tau)$, we write \widehat{T} for the tree obtained by attaching a new node labeled by the new symbol $\#$ to the root of T , i.e., $\widehat{T} := (1D, \widehat{\tau})$ with $\widehat{\tau}(\epsilon) := \#$ and, whenever $u \in D$, $\widehat{\tau}(1u) := \tau(u)$. Given an unranked tree $\mathfrak{T} = (D_{\mathfrak{T}}, \tau_{\mathfrak{T}})$ satisfying the conditions in Lemma 4.2.3, we can inductively define a Ω -labeled Υ -tree H_v for every $v \in D_{\mathfrak{T}}$, where $\Omega := \Sigma_{\perp}^2 \cup \{\#\}$. We set $H_v := (\alpha_v \otimes \beta_v)[\widehat{H_{v1}}, \dots, \widehat{H_{vr_v}}]$. Note that H_v might be infinite for some $v \in D_{\mathfrak{T}}$. If $H_v = (D, \tau)$, we also denote by $\mathbf{full}(H_v)$ the full infinite tree (Υ^*, τ') such that if $u \in D$, then $\tau'(u) := \tau(u)$; if $u \notin D$, then $\tau'(u) := \perp$ where $\perp := \begin{bmatrix} \perp \\ \perp \end{bmatrix}$. In other words, the tree $\mathbf{full}(H_v)$ is the tree H_v made full by padding finite branches by \perp .

We now construct the NBTA $\mathcal{B} = (\Omega, U, \delta, q_0, F)$. The automaton \mathcal{B} accepts precisely all Ω -labeled full infinite binary tree $\mathbf{full}(H_{\epsilon})$, where H_{ϵ} is generated by some unranked tree \mathfrak{T} satisfying the conditions in Lemma 4.2.3. The construction is very similar to the word case. For notational convenience, we shall sketch it only in the case of $\Upsilon = \{1, 2\}$, though the construction extends to k -ary trees with precisely the same complexity. Let q_F^1 and q_F^2 be new states not in $Q_1 \cup Q_2$. Denote by U_1 (resp. U_2) the set $Q_1 \cup \{q_F^1\}$ (resp. $Q_2 \cup \{q_F^2\}$). We define

$$U := (U_1 \times U_2 \times U_2) \cup (U_1 \times U_2 \times U_1 \times U_2 \times U_2).$$

The start state is $q_0 := (q_0^1, q_0^2, q_0^2)$. The states in $(U_1 \times U_2 \times U_2)$ are meant to handle the cases when no $\#$ has thus far been seen by \mathcal{B} . On the other hand, when \mathcal{B} is in $U_1 \times U_2 \times U_1 \times U_2 \times U_2$, at least one $\#$ has been seen. We now formally define the transition function δ . We first define how \mathcal{B} behaves when it is in $U_1 \times U_2 \times U_2$.

Suppose that $q_1 \in Q_1$, and $q_2, q'_2 \in Q_2$. For all $a, b \in \Sigma$, we set

$$\delta((q_1, q_2, q'_2), \begin{bmatrix} a \\ b \end{bmatrix}) := \left\{ \begin{array}{l} ((q_{L1}, q_{L2}, q'_{L2}), (q_{R1}, q_{R2}, q'_{R2})) \mid \\ (q_{L1}, q_{R1}) \in \delta_1(q_1, b), \\ (q_{L2}, q_{R2}) \in \delta_2(q_2, \begin{bmatrix} a \\ b \end{bmatrix}), \\ (q'_{L2}, q'_{R2}) \in \delta_2(q'_2, \begin{bmatrix} b \\ b \end{bmatrix}) \end{array} \right\}.$$

For all $a \in \Sigma$, $q_1 \in F_1 \cup \{q_F^1\}$ and $q'_2 \in F_2 \cup \{q_F^2\}$, we set

$$\delta((q_1, q_2, q'_2), \begin{bmatrix} a \\ \perp \end{bmatrix}) := \{((q_F^1, q_{L2}, q_F^2), (q_F^1, q_{R2}, q_F^2)) : (q_{L2}, q_{R2}) \in \delta_2(q_2, \begin{bmatrix} a \\ \perp \end{bmatrix})\}$$

and, if $q_2 \in F_2 \cup \{q_F^2\}$, we set

$$\delta((q_1, q_2, q'_2), \perp) := \{((q_F^1, q_F^2, q_F^2), (q_F^1, q_F^2, q_F^2))\}.$$

Remark 4.2.1 Observe that, for all $a, b \in \Sigma$, we have $\delta((q_1, q_2, q'_2), \begin{bmatrix} a \\ b \end{bmatrix}) = \emptyset$ if at least one of the following holds: $q_1 = q_F^1$, $q_2 = q_F^2$, or $q'_2 = q_F^2$. Similarly, for $a \in \Sigma$, we have $\delta((q_1, q_2, q'_2), \begin{bmatrix} a \\ \perp \end{bmatrix}) = \emptyset$ unless $q_1 = q_F^1$ and $q'_2 = q_F^2$. Likewise, we have $\delta((q_F^1, q_F^2, q_F^2), \begin{bmatrix} a \\ \perp \end{bmatrix}) = \emptyset$ unless $a = \perp$. This means that once \mathcal{B} is in (q_F^1, q_F^2, q_F^2) , it is “trapped” and is forced to only see the node label \perp . ■

Suppose now that $q_1 \in U_1 \setminus \{q_F^1\}$ and $q_2, q'_2 \in U_2 \setminus \{q_F^2\}$. We then set

$$\delta((q_1, q_2, q'_2), \#) := ((q_1, q'_2, q_1, q_2, q'_2), (q_F^1, q_F^2, q_F^2)).$$

Notice that the state sent to the right child is (q_F^1, q_F^2, q_F^2) as the right child of every $\#$ -labeled node in $\mathbf{full}(H_\epsilon)$ is \perp -labeled.

We now proceed with our definition of δ when \mathcal{B} is in $U_1 \times U_2 \times U_1 \times U_2 \times U_2$. Suppose that $q_1, q'_1 \in Q_1$ and $q_2, q'_2, q''_2 \in Q_2$. For all $a, b \in \Sigma$, we define

$$\delta((q_1, q_2, q'_1, q'_2, q''_2), \begin{bmatrix} a \\ b \end{bmatrix})$$

as

$$\left\{ ((q_{L1}, q_{L2}, q'_{L1}, q'_{L2}, q''_{L2}), (q_{R1}, q_{R2}, q'_{R1}, q'_{R2}, q''_{R2})) \mid \begin{array}{l} (q_{L1}, q_{R1}) \in \delta_1(q_1, b), \\ (q_{L2}, q_{R2}) \in \delta_2(q_2, \begin{bmatrix} a \\ b \end{bmatrix}), \\ (q'_{L1}, q'_{R1}) \in \delta_1(q'_1, a), \\ (q'_{L2}, q'_{R2}) \in \delta_2(q'_2, \begin{bmatrix} \perp \\ a \end{bmatrix}), \\ (q''_{L2}, q''_{R2}) \in \delta_2(q''_2, \begin{bmatrix} b \\ b \end{bmatrix}) \end{array} \right\}$$

If $a \in \Sigma$, $q_1 \in F_1$, and $q_2'' \in F_2$, we define

$$\delta((q_1, q_2, q_1', q_2', q_2''), \begin{bmatrix} a \\ \perp \end{bmatrix})$$

as

$$\left\{ ((q_F^1, q_{L2}, q_{L1}', q_{L2}', q_F^2), (q_F^1, q_{R2}, q_{R1}', q_{R2}', q_F^2)) \mid \begin{array}{l} (q_{L2}, q_{R2}) \in \delta_2(q_2, \begin{bmatrix} a \\ \perp \end{bmatrix}), \\ (q_{L1}', q_{R1}') \in \delta_1(q_1', a), \\ (q_{L2}', q_{R2}') \in \delta_2(q_2', \begin{bmatrix} \perp \\ a \end{bmatrix}) \end{array} \right\}$$

If $q_1, q_1' \in F_1$ and $q_2, q_2', q_2'' \in F_2$, we set $\delta((q_1, q_2, q_1', q_2', q_2''), \perp) := (q_F^1, q_F^2, q_F^1, q_F^2, q_F^2)$.

Finally, if $(q_1, q_2, q_1', q_2', q_2'') \in Q_1 \times Q_2 \times F_1 \times F_2 \times Q_2$, then we set

$$\delta((q_1, q_2, q_1', q_2', q_2''), l) := \begin{cases} \{(q_1, q_2'', q_1, q_2, q_2'')\} & \text{if } l = \#, \\ \emptyset & \text{otherwise.} \end{cases}$$

We now set

$$F := \{(q_F^1, q_F^2, q_F^2), (q_F^1, q_F^2, q_F^1, q_F^2, q_F^2)\} \cup Q_1 \times Q_2 \times F_1 \times F_2 \times Q_2$$

It is easy to see that \mathcal{B} recognizes precisely all trees $\mathbf{full}(H_\varepsilon)$, where H_ε is generated by some unranked tree \mathfrak{T} satisfying lemma 4.2.3. Furthermore, checking the definition of the transition function δ of \mathcal{B} , it is easy to see that $\|\mathcal{B}\| = O(\|\mathcal{A}\|^2 \times \|\mathcal{R}\|^3)$ and that the construction of \mathcal{B} takes time $O(|\Sigma|^2 \times \|\mathcal{A}\|^2 \times \|\mathcal{R}\|^3)$.

We now show how to construct from \mathcal{B} the automaton $\mathcal{A}' = (\Sigma, Q', \delta', q_0', F')$ that recognizes $\text{Rec}(\mathcal{L}(\mathcal{A}))$. The intuitive idea is similar to the word case: given a tree T , the automaton \mathcal{A}' guesses a tree $\mathbf{full}(H_\varepsilon)$, where H_ε is generated by an unranked tree $\mathfrak{T} = (D_{\mathfrak{T}}, \tau_{\mathfrak{T}})$ satisfying lemma 4.2.3 such that $\tau_{\mathfrak{T}}(\varepsilon) = (T \otimes \beta_\varepsilon[x_1, \dots, x_{r_\varepsilon}], q_0^1, q_0^2)$ for some context tree $\beta_\varepsilon[x_1, \dots, x_{r_\varepsilon}]$. More formally, we set $Q' := (U_1 \times U_2 \times U_2)$ and $q_0' = (q_0^1, q_0^2, q_0^2)$. The transition function is defined as follows:

$$\delta'((q_1, q_2, q_2'), a) = \bigcup_{b \in \Sigma_\perp} \delta((q_1, q_2, q_2'), \begin{bmatrix} a \\ b \end{bmatrix}).$$

Finally, we set

$$F' := \bigcup \{ (q_1, q_2, q_2') \in Q_1 \times Q_2 \times Q_2 \mid \mathcal{B}^{(q_1, q_2, q_2')} \text{ accepts some } \Omega\text{-labeled binary tree of the form } \widehat{T} \}.$$

Observe that F' can be computed by using the algorithm for checking emptiness for Büchi tree automata, which runs in quadratic time $O(|\mathcal{B}|^2) = O(|\Sigma|^2 \times \|\mathcal{A}\|^4 \times \|\mathcal{R}\|^6)$ (e.g. see [VW86a]). Finally, observe that $\|\mathcal{A}'\| = O(\|\mathcal{A}\| \times \|\mathcal{R}\|^2)$ and the total time taken to compute \mathcal{A}' is $O(|\Sigma|^2 \times \|\mathcal{A}\|^4 \times \|\mathcal{R}\|^6)$. Theorem 4.2.1 is now immediate.

Remark 4.2.2 Recall that the proof of the analogous statement for the word-automatic case (i.e. Lemma 4.1.3) could be greatly simplified by first determinizing the NWA \mathcal{R} at the cost of an exponential blow-up in the size of \mathcal{R} . Our proof for Lemma 4.1.3 provides an exponential reduction in the size of \mathcal{R} , although this is not necessary if complexity is not a primary concern. On the other hand, it is not obvious to adapt such a technique in the tree case (i.e. to prove Lemma 4.2.3). This is simply because non-deterministic top-down tree automata are not determinizable in general. Furthermore, replacing the NTA \mathcal{R} by a bottom-up deterministic automaton does not seem to help in the construction of the Büchi tree automaton \mathcal{B} since the standard definition of Büchi tree automata is top-down, which is more natural since the input tree is infinite. Incidentally, our proof technique above easily yields a generalization of the results from [KRS05] to the tree-automatic case. ■

4.2.4 An appetizer example

We now give an immediate application of Theorem 4.2.1 for deriving an optimal complexity (up to a polynomial) of checking recurrent reachability of regular ground tree rewrite systems. More concrete examples will be given in Chapter 6.

Regular ground tree rewrite systems

Recall that RGTRSs can be thought of as tree-automatic systems. We now give another proof of the result by Löding [Löd03, Löd06] on polynomial-time procedure for deciding recurrent reachability over RGTRSs. We first recall a classic result by Dauchet and Tison [DT90] (also see [CDG⁺07, Chapter 3]) on the reachability relation for RGTRSs.

Proposition 4.2.5 *The reachability relations of RGTRSs are effectively tree-regular relations. Furthermore, they can be computed in time polynomial in the size of the input RGTRS.*

The proof for the above proposition first constructs “ground tree transducers”, which can then be easily converted into a tree-automatic presentation (e.g. see [CDG⁺07,

Chapter 3]). Since reachability relations and strict reachability relations are polynomially interdefinable for tree-automatic systems (see Example 3.2.5), Theorem 4.2.1 and Proposition 4.2.5 gives the following immediate corollary.

Corollary 4.2.6 *Recurrent reachability over RGTRSs is solvable in polynomial-time. Furthermore, the set of configurations $\text{Rec}(\mathcal{L}(\mathcal{A}))$ which satisfies the recurrent reachability property is also regular for which an NTA can be computed in polynomial-time.*

4.3 Generalized Büchi conditions

We now consider a more general version of recurrent reachability. Given a transition system $\mathfrak{S} = \langle S, \{\rightarrow_a\}_{a \in \text{ACT}} \rangle$ and the sets $S_1, \dots, S_n \subseteq S$, we define $\text{Rec}(S_1, \dots, S_n)$ to be the set of all $s_0 \in S$ from which there exists an infinite path $\pi = s_0 s_1 \dots$ such that, for each $i = 1, \dots, n$, we have $s_j \in S_i$ for infinitely many $j \in \mathbb{N}$. In other words, $\text{Rec}(S_1, \dots, S_n)$ contains all $s \in S$ from which there exists an infinite path which visits each S_i (for all $i = 1, \dots, n$) infinitely often. In analogy with finite automata over ω -words, one may call this problem *recurrent reachability with generalized Büchi condition*². Observe that this definition coincides with the definition of recurrent reachability in the case when $n = 1$. On the other hand, it is *not* necessarily the case that $\text{Rec}(S_1, \dots, S_n) = \text{Rec}(\bigcup_{i=1}^n S_i)$ in general since the latter only enforces the existence of path which visits *at least one* S_i infinitely often.

In this section, we shall apply Theorem 4.1.1 and Theorem 4.2.1 to show how to handle generalized Büchi conditions for word/tree automatic systems. In contrast to recurrent reachability (without generalized Büchi conditions), the time complexity of our algorithm becomes exponential in the number n of “target automata”, which we show to be optimal even for simpler classes of word/tree automatic presentations including pushdown systems and ground tree rewrite systems. As applications of our main results of this section, we derive algorithms with optimal complexity for solving recurrent reachability with generalized Büchi conditions over pushdown systems, prefix-recognizable systems, and ground-tree rewrite systems.

²Automata over ω -words with generalized Büchi conditions, which recognize precisely ω -regular languages, are well-studied (e.g. see [Wol00])

4.3.1 Word-automatic systems

Let us first start with word-automatic systems. The following theorem is an extension on Theorem 4.1.1 to recurrent reachability with generalized Büchi conditions.

Theorem 4.3.1 *Suppose that \mathcal{C} is a class of automatic systems closed under transitive closure. Then, given a presentation $\eta \in \mathcal{C}$ over Σ of an automatic system $\mathfrak{S}_\eta = \langle S, \{\rightarrow_a\}_{a \in \text{ACT}} \rangle$ and NWAs $\mathcal{A}_1, \dots, \mathcal{A}_n$ over Σ , the set $\text{Rec}(\mathcal{L}(\mathcal{A}_1), \dots, \mathcal{L}(\mathcal{A}_n))$ is regular.*

Moreover, if \mathcal{C} is effectively closed under transitive closure with an ETC-witness \mathcal{M} , then an NWA recognizing $\text{Rec}(\mathcal{L}(\mathcal{A}_1), \dots, \mathcal{L}(\mathcal{A}_n))$ of size $O(\|\mathcal{R}\|^{2n} \times \prod_{i=1}^n \|\mathcal{A}_i\|^2)$ in time $O(|\Sigma|^{n+1} \times \|\mathcal{R}\|^{3n} \times \prod_{i=1}^n \|\mathcal{A}_i\|^3)$.

Notice that if the number n of target automata is fixed, then the algorithm runs in polynomial time. We shall see in Proposition 4.3.4 that the complexity in the above algorithm cannot be substantially lowered.

We now proceed with the proof of Theorem 4.3.1. Observe that, for each $s_0 \in S$, it is the case that $s_0 \in \text{Rec}(\mathcal{L}(\mathcal{A}_1), \dots, \mathcal{L}(\mathcal{A}_n))$ iff there exists a sequence $\{s_i\}_{i \in \mathbb{N}}$ such that $s_i \rightarrow^+ s_{i+1}$ for each integer $i \in \mathbb{N}$, and $s_{nk+i} \in \mathcal{L}(\mathcal{A}_i)$ for each pair of integers $k \geq 0$ and $i \in [1, n]$. Therefore, let \mathcal{R} be the NWA over $\Sigma_\perp \times \Sigma_\perp$ that accepts precisely \rightarrow^+ . Define a new binary relation $\rightarrow_1 \subseteq S \times S$, where for each $s_0, s_n \in S$

$$s_0 \rightarrow_1 s_n \Leftrightarrow \exists s_1, s_2, \dots, s_{n-1} \in S \left(\bigwedge_{i=0}^{n-1} (s_i \rightarrow^+ s_{i+1}) \wedge \bigwedge_{i=1}^n s_i \in \mathcal{L}(\mathcal{A}_i) \right).$$

By transitivity of \rightarrow^+ , we see that \rightarrow_1 is also a transitive relation. Furthermore, it is clear that, for each $s \in S$,

$$s \in \text{Rec}(\mathcal{L}(\mathcal{A}_1), \dots, \mathcal{L}(\mathcal{A}_n))[\rightarrow^+] \Leftrightarrow s \in \text{Rec}(\Sigma^*)[\rightarrow_1].$$

By Proposition 3.1.2, the relation \rightarrow_1 is regular for which an NWA of size $O(\|\mathcal{R}\|^n \times \prod_{i=1}^n \|\mathcal{A}_i\|)$ can be computed in time $O(|\Sigma|^n + 1 \times \|\mathcal{R}\|^n \times \prod_{i=1}^n \|\mathcal{A}_i\|)$. By Theorem 4.1.1, we can compute an NWA of size $O(\|\mathcal{R}\|^{2n} \times \prod_{i=1}^n \|\mathcal{A}_i\|^2)$ for the set

$$\text{Rec}(\mathcal{L}(\mathcal{A}_1), \dots, \mathcal{L}(\mathcal{A}_n))[\rightarrow^+]$$

in time $O(|\Sigma|^{n+1} \times \|\mathcal{R}\|^{3n} \times \prod_{i=1}^n \|\mathcal{A}_i\|^3)$.

4.3.2 Tree-automatic systems

We now proceed to the tree analogue of Theorem 4.3.1.

Theorem 4.3.2 *Suppose that \mathcal{C} is a class of tree-automatic systems closed under transitive closure. Then, given a presentation $\eta \in \mathcal{C}$ over Σ of a $\text{TREE}_k(\Sigma)$ -automatic system $\mathfrak{S}_\eta = \langle S, \{\rightarrow_a\}_{a \in \text{ACT}} \rangle$ and NTAs $\mathcal{A}_1, \dots, \mathcal{A}_n$ over $\text{TREE}_k(\Sigma)$, the set*

$$\text{Rec}(\mathcal{L}(\mathcal{A}_1), \dots, \mathcal{L}(\mathcal{A}_n))$$

is tree-regular. Moreover, if \mathcal{C} is effectively closed under transitive closure with an ETC-witness \mathcal{M} , then an NTA recognizing $\text{Rec}(\mathcal{L}(\mathcal{A}_1), \dots, \mathcal{L}(\mathcal{A}_n))$ of size $O(\|\mathcal{R}\|^{2n} \times \prod_{i=1}^n \|\mathcal{A}_i\|^2)$ in time $O(|\Sigma|^{n+1} \times \|\mathcal{R}\|^{6n} \times \prod_{i=1}^n \|\mathcal{A}_i\|^6)$.

The proof of this theorem is identical to the proof of Theorem 4.3.1 (except for using the tree analogues of the results for word-automatic systems), and hence is omitted. We shall see in Proposition 4.3.7 that the complexity in the above result cannot be substantially improved.

4.3.3 Applications

We now apply Theorem 4.3.1 and Theorem 4.3.2 for deriving optimal algorithms for recurrent reachability with generalized Büchi conditions over pushdown systems, prefix-recognizable systems, and RGTRSs.

Pushdown systems and prefix-recognizable systems

An immediate application of Theorem 4.3.1 and Proposition 4.1.7 is the following Theorem.

Theorem 4.3.3 *Recurrent reachability with generalized Büchi conditions expressed as NWAs $\mathcal{A}_1, \dots, \mathcal{A}_n$ (over the appropriate alphabet) is solvable in exponential time over pushdown systems. Furthermore, when the number n is fixed, then the problem can be solved in polynomial time.*

This result is now new, e.g., it can be derived using the techniques from [EKS03]. It turns out that the complexity cannot be substantially lowered, as the following proposition shows.

Proposition 4.3.4 *The problem of checking recurrent reachability with generalized Büchi conditions for the class of pushdown systems is PSPACE-hard.*

The proof of this proposition, which can be found in the appendix, is via a simple polynomial-time reduction from the emptiness of language intersections of DWAs,

which is PSPACE-complete [GJ79]. This also shows that the time complexity for Theorem 4.3.1. In fact, combining this proof with Birget's [Bir92] lower bound for the smallest size of NWAs recognizing the intersections of n languages of DWAs, it follows that the size of the NWA for $\text{Rec}(\mathcal{L}(\mathcal{A}_1), \dots, \mathcal{L}(\mathcal{A}_n))$ from Theorem 4.3.1 cannot be substantially lowered for pushdown systems.

Another application of Theorem 4.3.1 is the following result for prefix-recognizable systems.

Theorem 4.3.5 *Recurrent reachability with generalized Büchi conditions over prefix-recognizable systems is EXP-complete.*

As we saw earlier, the problem was already EXP-complete without generalized Büchi conditions due to Göller's recent EXP-completeness result for reachability for prefix-recognizable systems [Göl08]. This theorem can also be alternatively derived using the technique from [EKS03, KPV02].

Regular ground tree rewrite systems

We now apply Theorem 4.3.2 to answer Löding's open question [Löd06] regarding recurrent reachability with generalized Büchi conditions for regular ground tree rewrite systems (it was also not known to be decidable even for ground tree rewrite systems).

Theorem 4.3.6 *Recurrent reachability with generalized Büchi conditions expressed as the NTAs $\mathcal{A}_1, \dots, \mathcal{A}_n$ for RGTRSs is solvable in exponential time. Furthermore, the problem is solvable in polynomial time if n is fixed.*

It turns out this upper bound is essentially tight even for ground tree rewrite systems, as the following proposition shows.

Proposition 4.3.7 *Recurrent reachability with generalized Büchi conditions for GTRSs is EXP-hard.*

The proof of this proposition, which can be found in the appendix, is via a simple reduction from the nonemptiness problem for the intersections of the languages of n NTAs, which is EXP-complete [CDG⁺07].

4.4 Recurrent reachability via approximation

The transitive closure \rightarrow^+ of a regular relation \rightarrow is in general non-recursive (Proposition 3.1.5). In the previous sections, we have shown that recurrent reachability could be solved when \rightarrow^+ is effectively regular. In practice, we cannot always hope to be able to obtain an NWA for \rightarrow^+ since deciding whether the transitive closure \rightarrow^+ is regular for a given regular relation is already undecidable. In this section, we briefly consider the scenario when we have obtained an NWA representing an “under/upper approximations” of a regular relation \rightarrow . This scenario is reasonable since semi-algorithms that aim to compute under/upper approximations of transitive closure relations have been developed in the area of regular model checking (e.g. see [AJNS04, BLW03, Bou01, DLS02, Nil05]). More precisely, suppose that we are given a regular relation $R \supseteq \rightarrow^+$, i.e., a regular relation that overapproximates the real transitive closure relation. What can we say about the original recurrent reachability problem? Similarly, we may ask the same question when we are instead given a regular relation $R \subseteq \rightarrow^+$, i.e., a regular relation that underapproximates the real transitive closure relation. Note that R may not necessarily be transitive. In both cases, our techniques in the previous sections can be easily adapted to give partial answers for the original recurrent reachability problem over the original word/tree automatic transition system. We shall first state the result for the word-automatic case.

Theorem 4.4.1 *Given an NWA \mathcal{A} over Σ and a presentation η of a Σ^* -automatic system $\mathfrak{S}_\eta = \langle S, \{\rightarrow_a\}_{a \in \text{ACT}} \rangle$, suppose that \rightarrow^+ is the strict reachability relation of \mathfrak{S}_η and R is a regular relation, given as an NWA \mathcal{R} , satisfying $R \supseteq \rightarrow^+$ (resp. $R \subseteq \rightarrow^+$). Then, given a word $v_0 \in S$, we may check whether $v_0 \in \text{Rec}(\mathcal{L}(\mathcal{A}))[R]$ in time $O(|\Sigma|^2 \times 2^{O(\|\mathcal{R}\|)} \times \|\mathcal{A}\|^2)$ and, whenever $v_0 \notin \text{Rec}(\mathcal{L}(\mathcal{A}))[R]$ (resp. $v_0 \in \text{Rec}(\mathcal{L}(\mathcal{A}))[R]$) it is the case that $v_0 \notin \text{Rec}(\mathcal{L}(\mathcal{A}))[\rightarrow^+]$ (resp. $v_0 \in \text{Rec}(\mathcal{L}(\mathcal{A}))[\rightarrow^+]$).*

Recall from Section 2.5 that $\text{Rec}(\mathcal{L}(\mathcal{A}))[R]$ is defined even when the relation R is non-transitive. In other words, whenever R is an upper-approximation of \rightarrow^+ , we have sound negative tests for $v_0 \in \text{Rec}(\mathcal{L}(\mathcal{A}))[\rightarrow^+]$. In the case when R is an under-approximation of \rightarrow^+ , we have sound positive tests for $v_0 \in \text{Rec}(\mathcal{L}(\mathcal{A}))[\rightarrow^+]$. These simply follow from the simple observations that:

$$\begin{aligned} R \supseteq \rightarrow^+ &\quad \Rightarrow \quad \text{Rec}(\mathcal{L}(\mathcal{A}))[R] \supseteq \text{Rec}(\mathcal{L}(\mathcal{A}))[\rightarrow^+], \\ R \subseteq \rightarrow^+ &\quad \Rightarrow \quad \text{Rec}(\mathcal{L}(\mathcal{A}))[R] \subseteq \text{Rec}(\mathcal{L}(\mathcal{A}))[\rightarrow^+]. \end{aligned}$$

These observations hold even when R is not transitive. In the case when R is transitive, Theorem 4.4.1 then follows from Theorem 4.1.1, in which case better computational complexity that was stated in Theorem 4.4.1 is achievable (i.e. polynomial also in $\|\mathcal{R}\|$).

In the case when R is not transitive, it is also not difficult to prove Theorem 4.4.1. For this, we will have to go through the proof of Theorem 4.1.1. Recall that in the proof of Theorem 4.1.1 we first divided the set $\text{Rec}(\mathcal{L}(\mathcal{A}))$ into two sets $\text{Rec}_{\circlearrowleft}(\mathcal{L}(\mathcal{A}))$ and $\text{Rec}_{\rightarrow}(\mathcal{L}(\mathcal{A}))$. We may generalize the definitions of these two sets even when R is not transitive in the obvious way. More precisely, we define $v_0 \in \text{Rec}_{\circlearrowleft}(\mathcal{L}(\mathcal{A}))[R]$ iff there exists a sequence $\{v_i\}_{i \in \mathbb{N}}$ such that (1) $v_i \otimes v_j \in R$, for all integers $j > i \geq 0$, (2) $v_i \in \mathcal{L}(\mathcal{A})$ for all integer $i > 0$, and (3) $v_i = v_j$ for some $j > i \geq 0$. Similarly, we define $v_0 \in \text{Rec}_{\rightarrow}(\mathcal{L}(\mathcal{A}))[R]$ iff there exists a sequence $\{v_i\}_{i \in \mathbb{N}}$ such that (1) $v_i \otimes v_j \in R$, for all integers $j > i \geq 0$, (2) $v_i \in \mathcal{L}(\mathcal{A})$ for all integer $i > 0$, and (3) $v_i \neq v_j$ for all $j > i \geq 0$. We may compute an NWA for $\text{Rec}_{\circlearrowleft}(\mathcal{L}(\mathcal{A}))[R]$ in the same way we proved Lemma 4.1.2. In the case of $\text{Rec}_{\rightarrow}(\mathcal{L}(\mathcal{A}))[R]$, a proof that is similar to our proof of Lemma 4.1.3 can also be given. Recall that it suffices to consider only witnessing infinite sequences of words with strictly increasing lengths (see Figure 4.2). Now observe that the conditions on the runs of \mathcal{R} in Lemma 4.1.4 assume that $\mathcal{L}(\mathcal{R})$ is transitive (see Condition 4). This can however be easily fixed by considering each pair $(\alpha_i, \beta_i \beta_{i+1} \dots \beta_{j-1} \alpha_j)$ of suffixes in the witnessing sequence, i.e., by asserting that \mathcal{R} accepts $\alpha_i \otimes \beta_i \beta_{i+1} \dots \beta_{j-1} \alpha_j$ instead of only $\alpha_i \otimes \beta_i \alpha_{i+1}$. In addition, we use the same definition of ω -chains and so Proposition 4.1.5 can be directly used. Now, in the construction of the NBWA \mathcal{B} , we will additionally have to make sure that, for all integers $j > i \geq 0$, the word $\varepsilon \otimes \beta_{i+1} \dots \beta_{j-1} \alpha_j$ is accepted by \mathcal{R} from an appropriate state. Since there are only finite many states in \mathcal{R} , we simply have to keep track of every possible subset of the states in \mathcal{R} to handle this causing an exponential blow-up in the number of states in \mathcal{R} .

We shall also state the tree analogue of Theorem 4.4.1, which can be proven in the same way.

Theorem 4.4.2 *Given an NTA \mathcal{A} over $\text{TREE}_k(\Sigma)$ and a presentation η of a $\text{TREE}_k(\Sigma)$ -automatic system $\mathfrak{S}_\eta = \langle S, \{\rightarrow_a\}_{a \in \text{ACT}} \rangle$, suppose that \rightarrow^+ is the strict reachability relation of \mathfrak{S}_η and R is a regular relation, given as an NTA \mathcal{R} , satisfying $R \supseteq \rightarrow^+$ (resp. $R \subseteq \rightarrow^+$). Then, given a tree $T_0 \in S$, we may check whether $T_0 \in \text{Rec}(\mathcal{L}(\mathcal{A}))[R]$ in time $O(|\Sigma|^2 \times 2^{O(\|\mathcal{R}\|)} \times \|\mathcal{A}\|^4)$ and, whenever $T_0 \notin \text{Rec}(\mathcal{L}(\mathcal{A}))[R]$ (resp. $T_0 \in$*

$Rec(\mathcal{L}(\mathcal{A}))[R])$ it is the case that $T_0 \notin Rec(\mathcal{L}(\mathcal{A}))[\rightarrow^+]$ (resp. $T_0 \in Rec(\mathcal{L}(\mathcal{A}))[\rightarrow^+]$).

Remark 4.4.1 The proof of Theorem 4.4.1 also yields a proof of that “Ramseyan quantifiers” preserve regularity over word-automatic structures, which was proven by Rubin [Rub08]. For the tree case, the proof of Theorem 4.4.2 yields a proof of the regularity-preserving property of Ramseyan quantifiers over tree-automatic structures. This has been independently observed by Kartzow and Kuske this year³.

³Private communication with Kartzow (2010)

Chapter 5

Algorithmic metatheorems for logic model checking

In the previous chapter, we proved algorithmic metatheorems for decidable recurrent reachability over word/tree automatic systems and show that they can be used for deriving uniform proofs of decidability (with optimal complexity) for various recurrent reachability problems for pushdown systems, prefix-recognizable systems, and regular ground tree rewrite systems. Although recurrent reachability by itself is a rather weak property, in this chapter we shall see that the algorithmic metatheorems from the previous chapter can be used to obtain algorithmic metatheorems for decidable model checking with respect to various logics including LTL (and fragments thereof) with “complex fairness constraints” and extensions of first-order logic. As we shall see later in this chapter, they can be used to obtain optimal model checking algorithms for pushdown systems, prefix-recognizable systems, and regular ground tree rewrite systems (more applications can be found in the next chapter).

In Section 5.1, we study the problem of *LTL model checking over word/tree automatic systems with regular fairness constraints*: given a presentation η of a Σ^* -automatic (resp. $\text{TREE}_k(\Sigma)$ -automatic) system $\mathfrak{S}_\eta = \langle S, \{\rightarrow_a\}_{a \in \text{ACT}} \rangle$, a configuration $s_0 \in S$, an LTL formula φ over ACT, and a “fairness” NWA (resp. NTA) \mathcal{A} over Σ (resp. $\text{TREE}_k(\Sigma)$), decide whether, for each infinite path

$$\pi := s_0 \rightarrow_{a_1} s_1 \rightarrow_{a_2} s_2 \rightarrow_{a_3} \dots$$

in \mathfrak{S}_η satisfying $s_i \in \mathcal{L}(\mathcal{A})$ for infinitely many indices $i \in \mathbb{N}$, it is the case that $\pi \models \varphi$. When $\langle \eta, s_0, \varphi, \mathcal{A} \rangle$ is a positive instance of the problem, we write $(\mathfrak{S}_\eta, s_0, \mathcal{L}(\mathcal{A})) \models \varphi$ and say that \mathfrak{S}_η *satisfies* φ from s_0 with fairness constraint \mathcal{A} . Fairness constraints are

natural conditions which allow the users of a model checker to specify which paths clearly *cannot* occur in the actual systems being modeled, i.e., occur only in the *abstract* models. Only *fair* paths then should be considered by the model checker (see [BBF⁺01] for a more thorough discussion). As we shall see later, regular fairness constraints are powerful enough for modeling interesting fairness constraints. Unlike the case of recurrent reachability properties, the condition of effective closure under transitive closure on a class C of automatic systems is *not* sufficient to imply decidability of LTL model checking over C even without the extra fairness constraint. On the other hand, if we impose an extra condition that the class C is *closed under products with finite systems*, decidability of LTL model checking with regular fairness constraints over C can be retained. The time complexity of the algorithm is exponential in the size of the formula φ and polynomial in the size of the presentation η of the system and the initial configuration s_0 , assuming an oracle for computing the reachability relations. We shall also present an extension of this metatheorem to the problem of LTL model checking with *multi-regular* fairness constraints (i.e. when we have several regular constraints $\mathcal{A}_1, \dots, \mathcal{A}_n$ akin to generalized Büchi conditions), in which case we obtain the same complexity as the single-regular constraint case but exponential in the number n of regular constraints. We shall see in Section 5.4 that the condition of closure under products with finite systems is satisfied by the class of pushdown systems and prefix-recognizable systems, which yields decidability (with optimal complexity) of LTL with multi-regular fairness constraints over pushdown systems and prefix-recognizable systems.

Closure under products with finite systems is a rather strong condition, which is not satisfied by many classes of infinite-state systems (e.g. ground-tree rewrite systems). Such classes of infinite-state systems often have undecidable LTL model checking. In Section 5.2, we propose a weakening of the condition of closure under products with finite systems that could still be used to obtain algorithmic metatheorems of *fragments* of LTL with decidable model checking over word/tree automatic systems. This relaxed condition is called *closure under taking “subsystems”*: a class C of presentations of word/tree automatic systems is *closed under taking subsystems* if $\eta = \langle \mathcal{A}_S; \{\mathcal{A}_a\}_{a \in \text{ACT}} \rangle \in C$ and $\text{ACT}' \subseteq \text{ACT}$ implies that $\langle \mathcal{A}_S; \{\mathcal{A}_a\}_{a \in \text{ACT}'} \rangle \in C$. In other words, the *subsystem* of \mathfrak{S}_η that is obtained by removing some transition relations is still presented by some presentation in the class C . This is a rather innocuous condition which is satisfied by virtually every natural class of infinite-state systems that

is considered in the literature¹. We show that, when this condition is satisfied together with effective closure under transitive closure, two commonly considered fragments of LTL called $\text{LTL}(\mathbf{F}_s, \mathbf{G}_s)$ and LTL_{det} have decidable model checking with multi-regular fairness constraints. In this case, we show that these model checking problems have polynomial time data complexity (when the number of constraints is fixed), assuming an oracle for computing the reachability relations. We shall see in Section 5.4 that the class of RGTRSs is closed under taking subsystems and therefore have decidable $\text{LTL}(\mathbf{F}_s, \mathbf{G}_s)$ and LTL_{det} model checking with multi-regular fairness constraints (in fact, with polynomial-time data complexity). In addition, we show that multi-regular fairness constraints are sufficiently powerful for modelling natural fairness constraints when we use RGTRSs as abstract models of concurrent programs with an unbounded number of processes.

Finally, we conclude this chapter with a result which strongly suggests that obtaining algorithmic metatheorems for model checking branching-time logics over word or tree automatic transition systems with good computational complexity is difficult. More precisely, we show that model checking HM-logic (i.e. the simplest branching-time logic) is already nonelementary over a fixed word-automatic system.

Parts of the results in this chapter have previously appeared in [TL10] and [To09a].

5.1 Model checking LTL

In this section, we present our algorithmic metatheorems for decidable LTL model checking over word/tree automatic systems with regular fairness constraints. Observe first that this problem is more general than the problem of checking recurrent reachability since the latter can be easily reduced to the former: given a presentation η of a word/tree automatic system $\mathfrak{S}_\eta = \langle S, \{\rightarrow_a\}_{a \in \text{ACT}} \rangle$, an initial configuration $s_0 \in S$, and a target automaton \mathcal{A} , we have $s_0 \in \text{Rec}(\mathcal{L}(\mathcal{A}))$ iff it is not the case that $(\mathfrak{S}_\eta, s_0, \mathcal{L}(\mathcal{A})) \models \perp$. Therefore, a natural question is whether effective closure under transitive closure over a class \mathcal{C} is sufficient to ensure decidability of LTL model checking with regular fairness constraints over \mathcal{C} . It turns out that this is not the case even for LTL model checking without regular fairness constraints, as the following proposition shows.

Proposition 5.1.1 *There exists a presentation η of a fixed automatic system \mathfrak{S}_η such*

¹Similar remark has been made in [May01]

that:

1. $\{\eta\}$ is closed under transitive closure, but
2. LTL model checking over \mathfrak{S}_η is undecidable.

Proof. We shall give a reduction from the acceptance problem for the universal Turing machine. Fix a universal Turing machine $\mathcal{M} = (\Sigma, \Gamma, Q, \delta, q_0, q_F, \square)$ and its automatic presentation $\eta = \langle \mathcal{A}_S, \mathcal{A}_a \rangle$ of the transition system $\mathfrak{S}_\mathcal{M} = \langle S, \rightarrow_a \rangle$ generated by \mathcal{M} from Example 3.1.5, where $S = \Gamma^*(Q \times \Gamma)\Gamma^*$ and \rightarrow_a the one-step reachability relation of \mathcal{M} . Define a new transition relation $\rightarrow_{acc} \subseteq S \times S$ as follows: from any accepting configuration of the form $w(q_F, a)w'$, it is the case that $w(q_F, a)w' \rightarrow_{acc} w(q_F, a)w'$ (i.e. a self-loop). Observe that \rightarrow_{acc} is a regular relation. In addition, we introduce a “cheat” transition relation $\rightarrow_b \subseteq S \times S$ that can take any configuration $\mathbf{c} \in S$ to another configuration $\mathbf{c} \in S$, i.e., $\rightarrow_b = S \times S$. Observe that \rightarrow_b is a regular relation. Therefore, the transitive closure of $\rightarrow := \rightarrow_a \cup \rightarrow_{acc} \cup \rightarrow_b = \rightarrow_b$ is also regular. Let \mathcal{A}_{acc} and \mathcal{A}_b be NWAs that recognize, respectively, \rightarrow_{acc} and \rightarrow_b . Define a new automatic presentation $\eta' = \langle \mathcal{A}_S, \mathcal{A}_a, \mathcal{A}_b \rangle$ which generates the automatic transition system $\mathfrak{S}_{\eta'} := \langle S, \rightarrow_a, \rightarrow_{acc}, \rightarrow_b \rangle$. Therefore, the class $\{\eta'\}$ of automatic presentations is closed under transitive closure.

Now consider the LTL formula $\varphi := a \text{ U } acc$. It is easy to see that, for each $w \in \Sigma^*$, it is the case that $\mathfrak{S}_\eta, (q_0, \square)w \models \varphi$ iff w is accepted by \mathcal{M} . \square

We shall now introduce the condition of closure under products with finite systems for a class \mathcal{C} of word/tree automatic systems and show that decidability of LTL model checking with regular fairness constraints can be retained when this extra condition is imposed. Let us begin with word-automatic systems. Given a finite system $\mathfrak{F} = \langle Q, \{R_a\}_{a \in \text{ACT}} \rangle$ and an Σ^* -automatic system $\mathfrak{S}_\eta = \langle S, \{\rightarrow_a\}_{a \in \text{ACT}} \rangle$ presented by a presentation η , their *product* is the system $\mathfrak{F} \times \mathfrak{S}_\eta = \langle S', \{\rightarrow'_a\}_{a \in \text{ACT}} \rangle$ where

- S' is the language $QS = \{qv : q \in Q, v \in S\}$ over the alphabet $Q \cup \Sigma$, and
- \rightarrow'_a is such that $qv \rightarrow'_a pw$ iff $(p, q) \in R_a$ and $v \rightarrow_a w$.

Obviously, the system $\mathfrak{F} \times \mathfrak{S}_\eta$ is $(Q \cup \Sigma)^*$ -automatic, for which a presentation is computable in time $O(\|\mathfrak{F}\| \times \|\eta\|)$. Although there are non-unique presentations of the system $\mathfrak{F} \times \mathfrak{S}_\eta$, for convenience we will define the condition of closure under products with finite systems in such a way that the computation of the product system is efficient.

Definition 5.1.1 (Closure under products with finite systems) A class of automatic presentations \mathcal{C} is said to be (effectively) closed under products with finite systems if there exists an algorithm which, given a finite system \mathfrak{F} over ACT and an automatic transition system $\mathfrak{S}_\eta = \langle S, \{\rightarrow_a\}_{a \in ACT} \rangle$ presented by some $\eta \in \mathcal{C}$, computes a presentation $\eta' \in \mathcal{C}$ for the system $\mathfrak{F} \times \mathfrak{S}_\eta$ in time $O(\|\mathfrak{F}\| \times \|\eta\|)$.

We now state our algorithmic metatheorem for decidable LTL model checking with regular fairness constraints over word-automatic systems.

Theorem 5.1.2 Suppose that \mathcal{C} is a class of word-automatic presentations that are effectively closed under transitive closure with an ETC-witness \mathcal{M} and are closed under products with finite systems. Then, there exists an algorithm which, given a presentation η of a Σ^* -automatic system $\mathfrak{S}_\eta = \langle S, \{\rightarrow_a\}_{a \in ACT} \rangle$, a word $v_0 \in S$, an NWA \mathcal{A} over Σ , and an LTL formula ϕ over ACT , decides whether $(\mathfrak{S}_\eta, v_0, \mathcal{L}(\mathcal{A})) \models \phi$ in time linear in $|v_0|$ and polynomial in $\text{TIME}_{\mathcal{M}}(2^{O(\|\phi\|)} \times \|\eta\|)$ and $\|\mathcal{A}\|$.

Observe that when the formula ϕ is fixed, we obtain the same complexity as for our algorithmic metatheorem for recurrent reachability (up to a polynomial).

Proof. Let $\mathfrak{S}_\eta = \langle S, \{\rightarrow_a\}_{a \in ACT} \rangle$ be the system presented by the input automatic presentation η . We apply Vardi-Wolper's algorithm (i.e. Proposition 2.5.2) on the negation $\neg\phi$ of the given LTL formula ϕ yielding an NBWA $\mathcal{B} = (ACT, Q, \delta, q_0, F)$ of size $2^{O(\|\phi\|)}$ such that $\mathcal{L}(\mathcal{B}) = \llbracket \neg\phi \rrbracket$. The algorithm runs in time $2^{O(\|\phi\|)}$. It is clear that \mathcal{B} can be treated as a finite transition system (e.g. by omitting the initial state q_0 and set F of final states). We then use the assumption of closure under products with finite systems to obtain in time $O(\|\mathcal{B}\| \times \|\eta\|)$ an automatic presentation η' of the system $\mathcal{B} \times \mathfrak{S}_\eta = \langle QS, \{\rightarrow'_a\}_{a \in ACT} \rangle \in \mathcal{C}$. We may then apply the algorithm \mathcal{M} on η' to obtain the transitive closure \rightarrow^+ of $\bigcup_{a \in ACT} (\rightarrow'_a)$ in time

$$\text{TIME}_{\mathcal{M}}(\|\mathcal{B}\| \times \|\eta\|) = \text{TIME}_{\mathcal{M}}(2^{O(\|\phi\|)} \times \|\eta\|).$$

Then, for each $v_0 \in S$, we have $(\mathfrak{S}_\eta, v_0, \mathcal{L}(\mathcal{A})) \models \phi$ iff it is the case that there exists an infinite path

$$\pi := v_0 \rightarrow_{a_1} v_1 \rightarrow_{a_2} \dots$$

in \mathfrak{S}_η such that $v_i \in \mathcal{L}(\mathcal{A})$ for infinitely many indices $i \in \mathbb{N}$ and that $a_1 a_2 \dots \notin \llbracket \phi \rrbracket$ (or equivalently $a_1 a_2 \dots \in \mathcal{L}(\mathcal{B})$). The latter in turn is true iff it is the case that $v_0 \in \text{Rec}(FS, \mathcal{L}(\mathcal{A}))[\rightarrow^+]$. It is easy to see that an NWA \mathcal{N} for FS can be constructed in time $O(\|\mathcal{B}\| \times \|\eta\|)$. By Theorem 4.3.1, we may compute an NWA \mathcal{A}' recognizing the

set $\text{Rec}(FS, \mathcal{L}(\mathcal{A}))[\rightarrow^+]$ in time polynomial in $\text{TIME}_M(\|\mathcal{B}\| \times \|\eta\|)$ and $\|\mathcal{A}\|$. Testing whether $v_0 \in \mathcal{L}(\mathcal{A}')$ can then be done in $O((|\Sigma| + |Q|) \times |v_0| \times \|\mathcal{A}'\|)$, where $|Q| \leq \|\mathcal{B}\|$. This immediately implies the theorem. \square

This proof also shows that the decidability in Theorem 5.1.2 holds for any logic that can be converted to Büchi automata, e.g., Regular LTL and its extension with past operators [LS07, SL10]. Theorem 5.1.2 can also be extended to *LTL model checking with multi-regular constraints over word-automatic systems*: given a presentation η of a Σ^* -automatic system $\mathfrak{S}_\eta = \langle S, \{\rightarrow_a\}_{a \in \text{ACT}} \rangle$, a configuration $s_0 \in S$, an LTL formula ϕ over ACT, and a sequence of “fairness” NWAs $\mathcal{A}_1, \dots, \mathcal{A}_n$ over Σ , decide whether, for each infinite path

$$\pi := s_0 \rightarrow_{a_1} s_1 \rightarrow_{a_2} s_2 \rightarrow_{a_3} \dots$$

in \mathfrak{S}_η satisfying $\exists^\infty i (s_i \in \mathcal{L}(\mathcal{A}_j))$ for each $j \in \mathbb{N}$, it is the case that $\pi \models \phi$. When $\langle \eta, s_0, \phi, \{\mathcal{A}_i\}_{i=1}^n \rangle$ is a positive instance of the problem, we write $(\mathfrak{S}_\eta, s_0, \{\mathcal{L}(\mathcal{A}_i)\}_{i=1}^n) \models \phi$ and say that \mathfrak{S}_η satisfies ϕ from s_0 with fairness constraints $\mathcal{A}_1, \dots, \mathcal{A}_n$. This problem can be defined in a similar way for tree-automatic systems. The following theorem is now immediate from the proof of Theorem 5.1.2 and Theorem 4.3.1.

Theorem 5.1.3 *Suppose that \mathcal{C} is a class of word-automatic presentations that are effectively closed under transitive closure with an ETC-witness \mathcal{M} and are closed under products with finite systems. Then, there exists an algorithm which, given a presentation η of a Σ^* -automatic system $\mathfrak{S}_\eta = \langle S, \{\rightarrow_a\}_{a \in \text{ACT}} \rangle$, a word $v_0 \in S$, a sequence of NWAs $\{\mathcal{A}_i\}_{i=1}^n$ over Σ , and an LTL formula ϕ over ACT, decides whether $(\mathfrak{S}_\eta, v_0, \{\mathcal{L}(\mathcal{A}_i)\}_{i=1}^n) \models \phi$ in time linear in $|v_0|$ and polynomial in $\text{TIME}_M(2^{O(\|\phi\|)} \times \|\eta\|)$ and $\prod_{i=1}^n \|\mathcal{A}_i\|$.*

Observe that Theorem 5.1.2 is the restriction to Theorem 5.1.3 to $n = 1$. Also, observe that Theorem 5.1.3 when the formula ϕ is fixed, we obtain the same complexity as for recurrent reachability with generalized Büchi conditions (i.e. Theorem 4.3.1).

We now proceed to the tree case. Given a finite system $\mathfrak{F} = \langle Q, \{R_a\}_{a \in \text{ACT}} \rangle$ and an $\text{TREE}_k(\Sigma)$ -automatic system $\mathfrak{S}_\eta = \langle S, \{\rightarrow_a\}_{a \in \text{ACT}} \rangle$ presented by a presentation η , their *product* is the system $\mathfrak{F} \times \mathfrak{S}_\eta = \langle S', \{\rightarrow'_a\}_{a \in \text{ACT}} \rangle$ where the following conditions are satisfied:

- S' is the language containing trees $T = (D, \tau) \in \text{TREE}_k(\Sigma \cup Q)$ for which there is

a tree $T' = (D', \tau') \in \text{TREE}_k(\Sigma)$ and a state $q \in Q$ satisfying $D = 1D' \cup \{\varepsilon\}$ and

$$\tau(w) = \begin{cases} q & \text{if } w = \varepsilon, \\ \tau(1w) & \text{otherwise.} \end{cases}$$

For convenience, we shall write $T' = q(T)$ in the sequel.

- $q_1(T_1) \rightarrow'_a q_2(T_2)$ iff $(q_1, q_2) \in R_a$ and $T_1 \rightarrow_a T_2$.

Obviously, the system $\mathfrak{F} \times \mathfrak{S}_\eta$ is $\text{TREE}_k(\Sigma \cup Q)$ -automatic, for which a presentation is computable in time $O(\|\mathfrak{F}\| \times \|\eta\|)$. As for the word-automatic case, there are non-unique presentations for the system $\mathfrak{F} \times \mathfrak{S}_\eta$. For convenience, we will define the condition of closure under products with finite systems that ensures efficient computation of the product systems.

Definition 5.1.2 (Closure under products with finite systems; tree case) *A class C of tree-automatic presentations is said to be (effectively) closed under products with finite systems if there exists an algorithm which, given a tree-automatic transition system $\mathfrak{S}_\eta = \langle S, \{\rightarrow_a\}_{a \in \text{ACT}} \rangle$ presented by some $\eta \in C$ and a finite system \mathfrak{F} over ACT , computes a presentation $\eta' \in C$ for the system $\mathfrak{F} \times \mathfrak{S}_\eta$ in time $O(\|\mathfrak{F}\| \times \|\eta\|)$.*

We directly state our algorithmic metatheorem for decidable LTL model checking with multi-regular fairness constraints over tree-automatic systems.

Theorem 5.1.4 *Suppose that C is a class of tree-automatic presentations that are effectively closed under transitive closure with an ETC-witness \mathcal{M} and are closed under products with finite systems. Then, there exists an algorithm which, given a presentation η of a $\text{TREE}_k(\Sigma)$ -automatic system $\mathfrak{S}_\eta = \langle S, \{\rightarrow_a\}_{a \in \text{ACT}} \rangle$, a tree $T_0 \in S$, a sequence of NTAs $\{\mathcal{A}_i\}_{i=1}^n$ over $\text{TREE}_k(\Sigma)$, and an LTL formula φ over ACT , decides whether $(\mathfrak{S}_\eta, T_0, \{\mathcal{L}(\mathcal{A}_i)\}_{i=1}^n) \models \varphi$ in time linear in $|T_0|$ and polynomial in $\text{TIME}_{\mathcal{M}}(2^{O(\|\varphi\|)} \times \|\eta\|)$ and $\prod_{i=1}^n \|\mathcal{A}_i\|$.*

Observe that when the formula φ is fixed, we obtain the same complexity as for our algorithmic metatheorem for recurrent reachability with a generalized Büchi conditions (up to a polynomial). In addition, the subcase of LTL model checking with single-regular constraints can be obtained when n is restricted to 1. The proof of Theorem 5.1.4 is essentially identical to the proof of Theorem 5.1.3 and so is omitted. As in the word-automatic case, this theorem holds for any logic that can be converted to Büchi automata, e.g., Regular LTL and its extension with past operators [LS07, SL10].

5.2 Model checking LTL fragments

Closure under product with finite systems is a strong condition that is not satisfied by many classes of infinite-state systems (e.g. ground-tree rewrite systems). In this section, we study a weakening of the condition of closure under products with finite systems called *closure under taking subsystems*, and which fragments of LTL still have decidable model checking over word/tree automatic systems when this relaxed condition is imposed. Let us first start with the definition of this relaxed condition.

Definition 5.2.1 (Closure under taking subsystems) *A class of word/tree automatic presentations \mathcal{C} is said to be closed under taking subsystems if, given an automatic presentation $\langle \mathcal{A}_S, \{\mathcal{A}_a\}_{a \in ACT} \rangle \in \mathcal{C}$ and a subset $ACT' \subseteq ACT$, the automatic presentation $\langle \mathcal{A}_S, \{\mathcal{A}_a\}_{a \in ACT'} \rangle$ is also in \mathcal{C} .*

As we previously mentioned, this condition is rather weak and is satisfied by virtually every class of infinite-state systems that is considered in the literature. In this section, we shall show that this condition, combined with effective closure under transitive closure, is sufficient to guarantee decidability of two following fragments of LTL: LTL_{det} and $LTL(\mathbf{F}_s, \mathbf{G}_s)$. To this end, we shall first show an algorithmic metatheorem for decidable recurrent reachability checking on word/tree automatic transition systems with an extra *almost linear Büchi automaton* constraint.

5.2.1 Almost linear Büchi automata

As we saw from Proposition 2.5.2, LTL formulas can alternatively be represented as NBWAs. Model checking a given LTL formula then can be reduced to checking recurrent reachability of the original system with an additional NBWA constraint encoding the negation of the LTL formula. We shall now consider a subclass of NBWAs called *almost linear NBWAs* [BŘS09], which are sufficiently powerful to represent the negations of formulas in the fragments of LTL that we will consider in this section.

To define almost linear NBWAs, we shall first define the notion of linear NBWAs. An NBWA $\mathcal{A} = (\Sigma, Q, \delta, q_0, F)$ is called *linear* (a.k.a. *1-weak*) if there exists a partial order $\preceq \subseteq Q \times Q$ such that $q' \in \delta(q, a)$ implies $q \preceq q'$. Intuitively, the partial order ensures that once \mathcal{A} leaves a state q , it will never be able to come back to q . In other words, graph-theoretically \mathcal{A} looks like a dag possibly with self-loops, i.e., each strongly connected component (SCC) in \mathcal{A} contains only a single state. Observe that

every accepting run of \mathcal{A} must eventually self-loop in one final state $q \in F$, i.e., *sink* at q . In the sequel, the *depth* of \mathcal{A} refers to the length of the longest simple path in \mathcal{A} .

Definition 5.2.2 ([BRS09]) An almost linear NBWA \mathcal{A} over the alphabet Σ is a pair of a linear NBWA $\mathcal{B} = (\Sigma, Q, \delta, q_0, F)$ and a function χ mapping each final state $q \in F$ to an LTL formula over Σ of the form

$$\bigwedge_{i \in I} \mathbf{GF} p_i$$

where each p_i is a disjunction of positive atomic formulas. The language $\mathcal{L}(\mathcal{A})$ of \mathcal{A} contains all words $w \in \Sigma^\omega$ for which there is an accepting run of \mathcal{B} on w sinking at some $q \in F$ which satisfies $w \models \chi(q)$. The size $\|\mathcal{A}\|$ of \mathcal{A} is simply the sum of $\|\mathcal{B}\|$ and $\sum_{q \in F} \|\chi(q)\|$.

Almost linear NBWAs are not more powerful than NBWAs in terms of expressive power: there is a simple polynomial-time translation from almost linear NBWAs to NBWAs [BRS09] by a technique that is similar to the reduction from generalized Büchi automata to standard Büchi automata (cf. [Wol00]). We shall now prove a technical lemma that will be used to obtain algorithmic metatheorems for decidable model checking of LTL_{det} and $\text{LTL}(\mathbf{F}_s, \mathbf{G}_s)$ over word/tree automatic transition systems. First, let us fix the following notation: given a transition system $\mathfrak{S} = \langle S, \{\rightarrow_a\}_{a \in \text{ACT}} \rangle$ over ACT , a sequence of subsets $\sigma = \{S_i\}_{i=1}^n$ of S , and an almost linear NBWA $\mathcal{A} = (\text{ACT}, Q, \delta, q_0, F, \chi)$, write $\llbracket \mathcal{A} \rrbracket_{\mathfrak{S}, \sigma}^\exists$ to denote the set of all configurations $s_0 \in S$ from which there exists an infinite path

$$\pi = s_0 \rightarrow_{a_1} s_1 \rightarrow_{a_2} \dots$$

satisfying $a_1 a_2 \dots \in \mathcal{L}(\mathcal{A})$ and $\exists^\infty i (s_i \in S_j)$ for each $j \in [1, n]$.

Lemma 5.2.1 Suppose that \mathcal{C} is a class of word-automatic presentations that is effectively closed under transitive closure with an ETC witness \mathcal{M} and is closed under taking subsystems. Then, there exists an algorithm which, given a presentation η of a Σ^* -automatic system $\mathfrak{S}_\eta = \langle S, \{\rightarrow_a\}_{a \in \text{ACT}} \rangle$, a word $v_0 \in S$, a sequence of NWAs $\{\mathcal{A}_i\}_{i=1}^n$ over Σ , and an almost linear NBWA \mathcal{A} over ACT , decides whether $v_0 \in \llbracket \mathcal{A} \rrbracket_{\mathfrak{S}_\eta, \{\mathcal{L}(\mathcal{A}_i)\}_{i=1}^n}^\exists$ in time linear in $|v_0|$, polynomial in $\text{TIME}_{\mathcal{M}}(\|\eta\|)$, $\|\mathcal{A}\|$, and $\prod_{i=1}^n \|\mathcal{A}_i\|$, but exponential in the depth of \mathcal{A} .

Proof. Suppose that $\mathcal{A} = (\mathcal{B}, \chi)$, where $\mathcal{B} = (\text{ACT}, Q, \delta, q_0, F)$ is a linear NBWA over ACT . Let d denote the depth of \mathcal{A} . Loosely speaking, this lemma can be proven by

observing that each infinite path $s_0 \rightarrow_{a_1} s_1 \rightarrow_{a_2} \dots$ witnessing $v_0 \in \llbracket \mathcal{A} \rrbracket_{\mathfrak{S}_\eta, \{\mathcal{L}(\mathcal{A}_i)\}_{i=1}^n}^\exists$ can be divided into an initial finite segment $\{s_0\}_{i=1}^r$ and the remaining infinite segment $\{s_i\}_{i=r+1}^\omega$ such that there exists a sequence $\sigma = \{p_i\}_{i=1}^m$ of states in \mathcal{B} satisfying

- (1) $p_0 = q_0$,
- (2) $p_m \in F$,
- (3) for every $i \in [1, m]$ it is the case that $(p_i, a_i, p_{i+1}) \in \delta$ for some $a_i \in \text{ACT}$,
- (4) there is an accepting run $p_0^{j_0} \dots p_m^{j_m}$ (with each $j_i \in \mathbb{N}$) of \mathcal{B} (when viewed as automata over finite words) on $a_1 \dots a_r$,
- (5) $a_{r+1}a_{r+2} \dots \models \mathbf{G} \left(\bigwedge_{a: p_m \in \delta(p_m, a)} a \right) \wedge \chi(p_m)$, and
- (6) $\exists^\infty i \geq r+1 (s_i \in \mathcal{L}(\mathcal{A}_j))$ for each $j \in [1, n]$.

Conversely, given two such segments, we may glue them to obtain an infinite path witnessing $v_0 \in \llbracket \mathcal{A} \rrbracket_{\mathfrak{S}_\eta, \{\mathcal{L}(\mathcal{A}_i)\}_{i=1}^n}^\exists$. Our approach is to treat the finite segment and the infinite segments separately for each path σ of SCCs in \mathcal{B} .

Fix a path $\sigma = \{p_i\}_{i=1}^m$ of SCCs in \mathcal{B} satisfying the conditions (1)–(3). Of course, it is the case that m is at most the depth d of \mathcal{A} . Combining the assumption of closure under taking subsystems and effective closure under transitive closure of \mathcal{C} , given any subset $\text{ACT}' \subseteq \text{ACT}$, it is possible to compute NWAs $R_{\text{ACT}'}^+$ and $R_{\text{ACT}'}^*$ over Σ_\perp^2 for the transitive closure relations $\rightarrow_{\text{ACT}'}^+$ and $\rightarrow_{\text{ACT}'}^*$ for the transition system \mathfrak{S}_η . This can be done in time polynomial in $\text{TIME}_{\mathcal{M}}(\|\eta\|)$ for each given $\text{ACT}' \subseteq \text{ACT}$.

Consider now the relation $R_{\sigma,1} \subseteq S \times S$ containing tuples (s, s') for which there exists a path $s_0 \rightarrow_{b_1} \dots \rightarrow_{b_r} s_r$ such that $s_0 = s$, $s_r = s'$, and there exists a run of \mathcal{B} on $b_1 \dots b_r$ of the form $p_0^{j_0} \dots p_m^{j_m}$ for some $j_0, \dots, j_m \in \mathbb{N}$. It is easy to compute a conjunctive query $\phi_1(x, y)$ with at most $2m = O(d)$ quantifiers and $2m = O(d)$ conjuncts, each of the form $R_{\text{ACT}'}^+$ or $R_{\text{ACT}'}^*$, which expresses $R_{\sigma,1}$. That is, we will choose ACT' that expresses the set of labels a that takes p_i to p_{i+1} (or that self-loops on p_i). By Proposition 3.1.2, we obtain an NWA for $R_{\sigma,1}$ in time polynomial in $\text{TIME}_{\mathcal{M}}(\|\eta\|)$ and $\|\mathcal{A}\|$, but exponential in the depth d of \mathcal{A} .

We now consider the relation $R_{\sigma,2} \subseteq S$ consisting of elements $s \in S$ from which there exists an infinite path

$$s_0 \rightarrow_{a_1} s_1 \rightarrow_{a_2} \dots$$

such that $a_1 a_2 \dots \models \mathbf{G} \left(\bigwedge_{a: p_m \in \delta(p_m, a)} a \right) \wedge \chi(p_m)$, and $\exists^\infty i (s_i \in \mathcal{L}(\mathcal{A}_j))$ for each $j \in [1, n]$. Observe that it suffices to show that there exists an algorithm for computing an NWA

over Σ for S that runs in time polynomial in $\text{TIME}_{\mathcal{M}}(\|\eta\|)$, $\|\mathcal{A}\|$, and $\prod_{i=1}^n \|\mathcal{A}_i\|$, but exponential in the depth of \mathcal{A} ; for, if this is the case, we would be able to easily put together $R_{\sigma,1}$ and $R_{\sigma,2}$ by another conjunctive query with one conjunct and quantifier (and appealing to Proposition 3.1.2), which will finish the proof since there are at most $O(\|\mathcal{A}\|^d)$ paths of SCCs in \mathcal{A} satisfying (1)–(3).

Thus, it remains to show how to compute an NWA for $R_{\sigma,2}$. Suppose that $\chi(p_m) = \bigwedge_{i=1}^k \mathbf{GF} p_i$. Since p_i is a disjunction of positive atomic formulas, we may think of them as a subset of ACT. We consider a modified transition system $\mathfrak{S}' = \langle S', \{E_a\}_{a \in \text{ACT}} \rangle$ defined as follows:

- $S' = \{1, \dots, k\} \times S$,
- $(iv, jw) \in E_a$ iff $p_m \in \delta(p_m, a)$, $v \rightarrow_a w$, and whenever $i \neq j$, then $a \in p_i$.

It is easy to come up with a presentation η' in time polynomial in $\|\eta\|$ and $\|\mathcal{A}\|$ such that $\mathfrak{S}' = \mathfrak{S}_{\eta'}$. Suppose now that E^+ is the transitive closure of $\bigcup_{a \in \text{ACT}} E_a$. Observe now that

$$R_{\sigma,2} = \text{Rec}(\mathcal{L}(\mathcal{A}_1), \dots, \mathcal{L}(\mathcal{A}_n), \{1\} \times \Sigma^*, \dots, \{k\} \times \Sigma^*)[E^+].$$

Therefore, by Theorem 4.3.1, it suffices to show that an NWA for E^+ can be computed in time polynomial in $\|\mathcal{A}\|$ and $\text{TIME}_{\mathcal{M}}(\|\eta\|)$. To this end, observe first that for each path $(i_0, s_0) \rightarrow_{a_1} \dots \rightarrow_{a_r} (i_r, s_r)$ in \mathfrak{S}' , there exists a path from (i_0, s_0) to (i_r, s_r) of the form

$$(i_0, s_0) \rightarrow_{a_1} (i_0, s_1) \rightarrow_{a_2} \dots \rightarrow_{a_{r-1}} (i_0, s_{r-1}) \rightarrow_{a_r} (i_r, s_r).$$

In other words, to reach from (i, v) to (j, w) , we may always simply travel through only configurations of the form $\{i\} \times S$ and only at the end switch to (j, w) . Since $R_{\{a: p_m \in \delta(p_m, a)\}}^*$ can be computed in time polynomial in $\text{TIME}_{\mathcal{M}}(\|\eta\|)$, for each pair $(i, j) \in [1, k] \times [1, k]$, we may easily compute a conjunctive query $\psi_{(i,j)}(x, y)$ with one conjunct and one quantifier over the relations $\{R_{\text{ACT}'}^*, R_{\text{ACT}'}^+ : \text{ACT}' \subseteq \text{ACT}\}$ which expresses precisely all pairs in E^+ of the form (iv, jw) . The relation E^+ can then be easily obtained by taking union of $k^2 = O(\|\mathcal{A}\|)$ NWAs which represent each $\llbracket \psi_{(i,j)} \rrbracket$.

□

We shall now state the tree analogue of Lemma 5.2.1. The proof is completely identical to the word case and therefore is omitted.

Lemma 5.2.2 *Suppose that \mathcal{C} is a class of tree-automatic presentations that is effectively closed under transitive closure with an ETC witness \mathcal{M} and is closed under taking subsystems. Then, there exists an algorithm which, given a presentation η of a $\text{TREE}_k(\Sigma)$ -automatic system $\mathfrak{S}_\eta = \langle S, \{\rightarrow_a\}_{a \in \text{ACT}} \rangle$, a tree $T_0 \in S$, a sequence of NTAs $\{\mathcal{A}_i\}_{i=1}^n$ over $\text{TREE}_k(\Sigma)$, and an almost linear NBWA \mathcal{A} over ACT , decides whether $T_0 \in \llbracket \mathcal{A} \rrbracket_{\mathfrak{S}_\eta, \{\mathcal{L}(\mathcal{A}_i)\}_{i=1}^n}^\exists$ in time linear in $|T_0|$, polynomial in $\text{TIME}_{\mathcal{M}}(\|\eta\|)$, $\|\mathcal{A}\|$, and $\prod_{i=1}^n \|\mathcal{A}_i\|$, but exponential in the depth of \mathcal{A} .*

5.2.2 LTL_{det} : Deterministic LTL

Deterministic LTL (or LTL_{det}) is logic proposed by Maidl [Mai00] that captures a common fragment of LTL and CTL. We start with the definition of the syntax of LTL_{det} :

$$\begin{aligned} \phi, \phi' \quad := \quad & p \mid \mathbf{X}\phi \mid \phi \wedge \phi' \mid (p \wedge \phi) \vee (\neg p \wedge \phi') \mid \\ & (p \wedge \phi) \mathbf{U}(\neg p \wedge \phi') \mid (p \wedge \phi) \mathbf{W}(\neg p \wedge \phi'). \end{aligned}$$

Here p is a boolean combination of ACT . The semantics can be defined in the same way as for LTL. For example, $\phi \mathbf{W} \phi'$ is interpreted as the formula $\mathbf{G}\phi \vee (\phi \mathbf{U} \phi')$, i.e., the *weak until* operator. Maidl [Mai00] showed that negations of LTL_{det} formulas can be translated into linear NBWAs efficiently, which is in contrast to the general LTL formulas.

Lemma 5.2.3 ([Mai00]) *There exists a polynomial-time algorithm which, given an LTL_{det} formula ϕ over Σ , computes a linear NBWA $\mathcal{A}_{\neg\phi}$ of size $O(\|\phi\|)$ such that $\mathcal{L}(\mathcal{A}_{\neg\phi}) = \llbracket \neg\phi \rrbracket$.*

To obtain our algorithmic metatheorems for LTL_{det} , we simply need to combine Maidl's result above with Lemma 5.2.1 or Lemma 5.2.2.

Theorem 5.2.4 *Suppose that \mathcal{C} is a class of word/tree automatic presentations that are effectively closed under transitive closure with an ETC-witness \mathcal{M} and are closed under taking subsystems. Then, there exists an algorithm which, given a presentation η of a Σ^* -automatic (resp. $\text{TREE}_k(\Sigma)$ -automatic) system $\mathfrak{S}_\eta = \langle S, \{\rightarrow_a\}_{a \in \text{ACT}} \rangle$, a configuration $s_0 \in S$, a sequence of NWAs (resp. NTAs) $\{\mathcal{A}_i\}_{i=1}^n$ over Σ (resp. over $\text{TREE}_k(\Sigma)$) and an LTL_{det} formula ϕ , decides whether $\mathfrak{S}_\eta, s_0, \{\mathcal{A}_i\}_{i=1}^n \models \phi$ in time linear in $|s_0|$, polynomial in $\text{TIME}_{\mathcal{M}}(\|\eta\|)$ and $\prod_{i=1}^n \|\mathcal{A}_i\|$, and exponential in $\|\phi\|$.*

In fact, if we also assume closure under products with finite systems, a better complexity can be obtained. The following theorem can be obtained by following the proofs

of Theorem 5.1.2 and replace the use of Vardi-Wolper's construction by Maidl's result above.

Theorem 5.2.5 *Suppose that \mathcal{C} is a class of word/tree automatic presentations that are effectively closed under transitive closure with an ETC-witness \mathcal{M} and are closed under products with finite systems. Then, there exists an algorithm which, given a presentation η of a Σ^* -automatic (resp. $\text{TREE}_k(\Sigma)$ -automatic) system $\mathfrak{S}_\eta = \langle S, \{\rightarrow_a\}_{a \in \text{ACT}} \rangle$, a configuration $s_0 \in S$, a sequence of NWAs (resp. NTAs) $\{\mathcal{A}_i\}_{i=1}^n$ over Σ (resp. $\text{TREE}_k(\Sigma)$) and an LTL_{det} formula ϕ over ACT , decides whether $\mathfrak{S}_\eta, s_0, \{\mathcal{A}_i\}_{i=1}^n \models \phi$ in time linear in $|s_0|$ and polynomial in $\text{TIME}_{\mathcal{M}}(\|\phi\| \times \|\eta\|)$ and $\prod_{i=1}^n \|\mathcal{A}_i\|$.*

5.2.3 $\text{LTL}(\mathbf{F}_s, \mathbf{G}_s)$: LTL with only strict future/global operators

We now proceed to the fragment $\text{LTL}(\mathbf{F}_s, \mathbf{G}_s)$ of LTL with only modalities \mathbf{F}_s and \mathbf{G}_s . This fragment is strictly more expressive than the LTL fragment with the non-strict versions \mathbf{F} and \mathbf{G} of the modalities \mathbf{F}_s and \mathbf{G}_s as the former can be expressed in terms of the latter, e.g., $\mathbf{F}\phi \equiv \phi \vee \mathbf{F}_s\phi$. Observe that the fragment $\text{LTL}(\mathbf{F}_s, \mathbf{G}_s)$ is closed under negation. We next recall a known translation from $\text{LTL}(\mathbf{F}_s, \mathbf{G}_s)$ formulas to almost linear NBWAs.

Lemma 5.2.6 ([Reh07, BŘS09]) *There exists a double-exponential time translation from $\text{LTL}(\mathbf{F}_s, \mathbf{G}_s)$ formulas ϕ over ACT to almost linear NBWAs \mathcal{A} over ACT such that $\llbracket \phi \rrbracket = \mathcal{L}(\mathcal{A})$. Furthermore, the depth of \mathcal{A} is exponential in $\|\phi\|$.*

It is presently open whether the double-exponential time upper bound from [Reh07, BŘS09] can be improved. To obtain our algorithmic metatheorems for $\text{LTL}(\mathbf{F}_s, \mathbf{G}_s)$, we simply need to combine this lemma with Lemma 5.2.1 and Lemma 5.2.2.

Theorem 5.2.7 *Suppose that \mathcal{C} is a class of word/tree automatic presentations that are effectively closed under transitive closure with an ETC-witness \mathcal{M} and are closed under taking subsystems. Then, there exists an algorithm which, given a presentation η of a Σ^* -automatic (resp. $\text{TREE}_k(\Sigma)$ -automatic) system $\mathfrak{S}_\eta = \langle S, \{\rightarrow_a\}_{a \in \text{ACT}} \rangle$, a configuration $s_0 \in S$, a sequence of NWAs (resp. NTAs) $\{\mathcal{A}_i\}_{i=1}^n$ over Σ (resp. over $\text{TREE}_k(\Sigma)$) and an $\text{LTL}(\mathbf{F}_s, \mathbf{G}_s)$ formula ϕ , decides whether $\mathfrak{S}_\eta, s_0, \{\mathcal{A}_i\}_{i=1}^n \models \phi$ in time linear in $|s_0|$, polynomial in $\text{TIME}_{\mathcal{M}}(\|\eta\|)$ and $\prod_{i=1}^n \|\mathcal{A}_i\|$, and double-exponential in $\|\phi\|$.*

The upper bound in terms of the size of the formula in this theorem seems not optimal. To improve this upper bound, one has to first answer whether the upper bound from Lemma 5.2.6 can be improved further, which we leave as an open problem.

5.3 Model checking $\text{FO}_{\text{REG}}(\text{Reach} + \text{EGF})$

In this section, we give algorithmic metatheorems for showing decidability of an extension of the logic $\text{FO}_{\text{REG}}(\text{Reach})$ and $\text{FO}(\text{Reach})$, called $\text{FO}_{\text{REG}}(\text{Reach} + \text{EGF})$ and $\text{FO}(\text{Reach} + \text{EGF})$, over word/tree automatic systems.

We define the logic $\text{FO}_{\text{REG}}(\text{Reach} + \text{EGF})$ as an extension of $\text{FO}_{\text{REG}}(\text{Reach})$ with the generalized recurrent reachability operator with ω -regular constraints on the way. More precisely, the logic $\text{FO}_{\text{REG}}(\text{Reach} + \text{EGF})$ over ACT can be defined as follows:

- Each $\text{FO}_{\text{REG}}(\text{Reach})$ formula over ACT is an $\text{FO}_{\text{REG}}(\text{Reach} + \text{EGF})$ formula over ACT.
- Whenever $\varphi_1, \dots, \varphi_n$ are each an $\text{FO}_{\text{REG}}(\text{Reach} + \text{EGF})$ formula over ACT with one free variable and \mathcal{B} is an NBWA over the alphabet ACT, then

$$\text{EGF}_{\mathcal{B}}(\varphi_1, \dots, \varphi_n)$$

is an $\text{FO}_{\text{REG}}(\text{Reach} + \text{EGF})$ formula over ACT with one free variable.

- The logic $\text{FO}_{\text{REG}}(\text{Reach} + \text{EGF})$ is closed under boolean combinations and first-order quantification, with the standard rules for free variables.

We only need to provide the semantics for the second rule (the rest is standard). Given a transition system $\mathfrak{S} = \langle S, \{\rightarrow_a\}_{a \in \text{ACT}} \rangle$ and $s \in S$, we define

- $\mathfrak{S} \models \text{EGF}_{\mathcal{B}}(\varphi_1, \dots, \varphi_n)(s)$ iff there exists an infinite path

$$s \rightarrow_{a_1} s_1 \rightarrow_{a_2} s_2 \rightarrow_{a_2} \dots$$

in \mathfrak{S} such that $a_1 a_2 a_3 \dots \in \mathcal{L}(\mathcal{B})$ and, for each $j = 1, \dots, n$, we have $s_i \models \varphi_j$ for infinitely many $i \in \mathbb{N}$.

Observe that the semantics of the operators $\text{EGF}_{\mathcal{B}}$ are similar to recurrent reachability with generalized Büchi conditions. We define $\text{FO}(\text{Reach} + \text{EGF})$ as a sublogic of $\text{FO}_{\text{REG}}(\text{Reach} + \text{EGF})$ containing formulas in which each occurrence of the reachability operator $\text{Reach}_{\mathcal{A}}(x, y)$ satisfies $\mathcal{L}(\mathcal{A}) = \Gamma^*$ for some $\Gamma \subseteq \text{ACT}$, and each occurrence of

the recurrent reachability operator $\text{EGF}_{\mathcal{B}}$ satisfies $\mathcal{L}(\mathcal{B}) = \Gamma^\omega$ for some $\Gamma \subseteq \text{ACT}$. We now state our algorithmic metatheorem for $\text{FO}_{\text{REG}}(\text{Reach} + \text{EGF})$.

Theorem 5.3.1 *Suppose that \mathcal{C} is a class of word/tree automatic systems that are effectively closed under transitive closure and closed under products with finite systems. Then, given an $\text{FO}_{\text{REG}}(\text{Reach} + \text{EGF})$ formula $\varphi(\bar{x})$ over ACT and a presentation $\eta \in \mathcal{C}$ of a word/tree automatic system $\mathfrak{S}_\eta = \langle S, \{\rightarrow_a\}_{a \in \text{ACT}} \rangle$, the set $\llbracket \varphi \rrbracket$ is effectively regular. That is, model checking $\text{FO}_{\text{REG}}(\text{Reach} + \text{EGF})$ over \mathcal{C} is decidable.*

The proof of this theorem can be easily done due to effective closures under first-order operations for word/tree automatic systems (Proposition 3.1.1 and Proposition 3.2.1). When seeing a predicate $\text{Reach}_{\mathcal{A}}(x, y)$, we first construct a product with the finite system \mathcal{A} and then apply the assumption of effective closure under transitive closure. Similarly, when we encounter a $\text{EGF}_{\mathcal{A}}$ operator, we first construct a product with finite system \mathcal{B} and then applying our algorithmic metatheorem for recurrent reachability with generalized Büchi conditions (Theorem 4.3.1 and Theorem 4.3.2). We may also relax the condition of closure under products with finite systems and instead use the much weaker condition of closure under taking subsystems. In this case, we obtain the following theorem using a similar proof.

Theorem 5.3.2 *Suppose that \mathcal{C} is a class of word/tree automatic systems that is effectively closed under transitive closure and closed under taking subsystems. Then, given a presentation $\eta \in \mathcal{C}$ of a word/tree automatic system $\mathfrak{S}_\eta = \langle S, \{\rightarrow_a\}_{a \in \text{ACT}} \rangle$ and an $\text{FO}(\text{Reach} + \text{EGF})$ formula $\varphi(\bar{x})$ over ACT , the set $\llbracket \varphi \rrbracket$ is effectively regular. That is, model checking $\text{FO}(\text{Reach} + \text{EGF})$ over \mathcal{C} is decidable.*

5.4 Several appetizer applications

In this section, we give applications of the algorithmic metatheorems that we proved in this chapter on pushdown systems, prefix-recognizable systems, and regular ground tree rewrite systems. More examples will be given in the next chapter.

5.4.1 Pushdown systems

Consider the problem of model checking LTL over PDSs. This combined complexity was initially shown to be EXP-complete by Bouajjani, Esparza, and Maler [BEM97].

The lower bound in [BEM97] can also be adapted to show EXP-hardness for expression complexity. On the other hand, the problem is solvable in P for a fixed LTL formula [BEM97]. These upper bounds also hold for LTL formulas with *regular valuations* [EKS03], which are powerful enough for encoding multi-regular fairness constraints. Different proofs for these upper bounds have also been given (e.g. [PV04]). Our techniques from the previous sections (Theorem 5.1.2 and Proposition 4.1.7) give yet another proof for these upper bounds since PDSs are easily seen to be closed under products with finite systems.

Proposition 5.4.1 *Model checking LTL over pushdown systems with multi-regular fairness constraints is in EXP. For a fixed formula and fixed number of regular constraints, the problem is solvable in P.*

In fact, using our techniques, better combined complexity can be immediately obtained for LTL_{det} . The following proposition easily follows from Theorem 5.2.5 and Proposition 4.1.7.

Proposition 5.4.2 *Model checking LTL_{det} over pushdown systems with a fixed number of regular constraints is in P.*

Our algorithmic metatheorems for $FO_{REG}(Reach + EGF)$ can also be used to give decidability of $FO_{REG}(Reach + EGF)$ model checking over PDSs, though this is rather immediate from the decidability of model checking over PDSs with respect to MSO formulas [MS85].

5.4.2 Prefix-recognizable systems

We now proceed to the problem of LTL model checking over prefix-recognizable systems with multi-regular fairness constraints. This problem is known to be EXP-complete, even for a fixed formula and no regular fairness constraints [KPV02]. Combining Theorem 5.1.2 and Proposition 4.1.11, we give yet another proof of this result since prefix-recognizable systems are easily seen to be closed under products with finite systems.

Proposition 5.4.3 *Model checking LTL with multi-regular fairness constraints over prefix-recognizable systems is solvable in EXP.*

On the other hand, our algorithmic metatheorem for LTL_{det} does not help lower the complexity of the problem since model checking a fixed LTL_{det} formula (with no regular fairness constraint) over prefix-recognizable system is already EXP-hard, which follows from the EXP-completeness of checking reachability over prefix-recognizable systems [Göl08]. In addition, our algorithmic metatheorems for $FO_{REG}(Reach + EGF)$ can also be used to give decidability of $FO_{REG}(Reach + EGF)$ model checking over prefix-recognizable systems, though this immediately follows from the decidability of model checking over prefix-recognizable systems with respect to MSO formulas [Cau03].

5.4.3 Regular ground tree rewrite systems

Let us now proceed to the problem of model checking RGTRSs. We first start with a negative result about model checking LTL over GTRSs.

Proposition 5.4.4 *The problem of model checking a fixed LTL formula over GTRSs (with no regular fairness constraints) is undecidable.*

This proposition can be easily proved by an easy adaptation of the proof of the undecidability of model checking LTL over PA-processes [BKRS09]. In fact, this undecidability result holds for the fragment of LTL only with operator **U**, or the fragment with only operator **F** and **X**. In addition, observe that the class of GTRSs (or RGTRSs) is not closed under product with finite systems, which explains why our algorithmic metatheorems for decidable LTL model checking over tree-automatic systems fail in this case. On the other hand, we can still recover some decidable fragments as the following theorem shows.

Theorem 5.4.5 *Model checking LTL_{det} with multi-regular fairness constraints over RGTRSs is in EXP. Model checking $LTL(\mathbf{F}_s, \mathbf{G}_s)$ with multi-regular fairness constraints over RGTRSs is solvable in double exponential time. Furthermore, for fixed formulas and a fixed number of regular fairness constraints, model checking LTL_{det} and $LTL(\mathbf{F}_s, \mathbf{G}_s)$ over RGTRSs is in P.*

This theorem is a simple corollary of Theorem 5.2.4, Theorem 5.2.7, and Proposition 4.2.5. In the case of non-fixed LTL_{det} or $LTL(\mathbf{F}_s, \mathbf{G}_s)$ formulas but a fixed number of regular fairness constraints, can the complexity of the problem be improved to P? For example, this is the case for recurrent reachability as we showed in Theorem 4.3.6.

This is unlikely to be the case for $LTL(\mathbf{F}_s, \mathbf{G}_s)$ since the problem is already coNP-complete even for finite systems [SC85]. However, LTL_{det} model checking over finite systems is solvable in P [Mai00]. Despite this, this is not the case for GTRSs even in the absence of regular fairness constraints.

Proposition 5.4.6 *Model checking LTL_{det} over GTRSs is coNP-hard.*

The proof of this proposition is by a simple reduction from the complement of the hamiltonian path problem, which is given in the appendix. We leave the precise combined complexity of LTL_{det} and $LTL(\mathbf{F}_s, \mathbf{G}_s)$ over GTRSs and RGTRSs for future work.

An application of Theorem 5.4.5 is for model checking concurrent programs with an unbounded number of processes in the presence of fairness constraints. GTRSs are natural models for modeling concurrent programs with an unbounded number of processes, but with only “local communications”. We may think of each node in a tree as a *process* in our concurrent programs. Trees ensure a hierarchical structure amongs the processes in the programs: a node v with children $v1, \dots, vk$ means that the processes $v1, \dots, vk$ are *subprocesses* of the *parent* process v . GTRSs rules ensure that communications only happen “locally”. Node labels in the trees then correspond to the finite abstract domains that are obtained by *predicate abstractions* [GS97] (in the manner of how pushdown systems can be obtained from sequential programs, c.f. [BMMR01, EK99]). A natural fairness constraint for concurrent programs is that there is each *leaf process* will eventually be executed (i.e. there is some rewrite rule that will be used to rewrite a subtree containing this node). This fairness constraint can be modeled using multi-regular fairness constraints as follows. First introduce two extra colors $\{1, 2\}$ for the leaf nodes, i.e., if Σ is the original node alphabet, we use $\Sigma' := \{1, 2, ?\} \times \Sigma$ for the new node alphabet. We then define a new GTRS which has the same rule as the original GTRS, but ensures that internal nodes are labeled by $\{?\} \times \Sigma$ while leaves are labeled by $\{1, 2\} \times \Sigma$. Each rule will then allow the color in the leaves to stay the same or toggle. We then simply have to consider infinite runs where the set of trees in which all the leaves are labeled by $\{1\} \times \Sigma$ and set of trees in which all the leaves are labeled by $\{2\} \times \Sigma$ are both visited infinitely often, which can be modeled using two regular fairness constraints.

We close this section by mentioning the application of Theorem 5.3.2 to RGTRSs.

Theorem 5.4.7 *Model checking $FO(\text{Reach} + \text{EGF})$ over RGTRSs is decidable.*

5.5 A nonelementary lower bound for HM-logic

In Section 5.3, we obtained algorithmic metatheorems for decidable model checking of $\text{FO}_{\text{REG}}(\text{Reach} + \text{EGF})$. The complexity that we obtained was nonelementary in the size of the formula, which was optimal since there exists a fixed pushdown system (i.e. the infinite binary tree) with a nonelementary $\text{FO}(\text{Reach})$ theory. A natural question is whether better upper bounds can be given for weaker logics, say, EF-logic. In this section, we shall prove that this is not the case even for HM-logic. More precisely, we shall show that there exists a fixed automatic transition system with a nonelementary HM-logic model checking. In fact, the transitive closure of the union of the transition relations in this system is also regular, which strongly suggests that any algorithmic metatheorem for decidable branching-time logics (like CTL and EF-logic) must impose much stronger restrictions for it to have nice computational complexity properties. Our nonelementary lower bound also strengthens Proposition 3.1.4 and answers an open question from [BG09] on the expression complexity of modal logic over rational graphs.

Theorem 5.5.1 ([To09a]) *There exists a fixed word-automatic transition system $\mathcal{T} = \langle S, \{\rightarrow_a\}_{a \in \text{ACT}} \rangle$ and a state s_0 in \mathcal{S} such that checking whether (\mathcal{S}, s_0) satisfies a given HM-logic formula is nonelementary. Furthermore, the transitive closure of $\bigcup_{a \in \text{ACT}} (\rightarrow_a)$ is regular.*

We shall now give a proof for this theorem. Recall that $\langle \{0, 1\}^*, \text{succ}_0, \text{succ}_1, \preceq \rangle$ is the infinite binary tree with a descendant relation. We shall start with an observation that the FO^4 theory of $\langle \{0, 1\}^*, \text{succ}_0, \text{succ}_1, \preceq \rangle$ is nonelementary.

Proposition 5.5.2 *The FO^4 theory of $\langle \{0, 1\}^*, \text{succ}_0, \text{succ}_1, \preceq \rangle$ is nonelementary.*

Although the first-order theory of $\langle \{0, 1\}^*, \text{succ}_0, \text{succ}_1, \preceq \rangle$ was proved to be nonelementary in [CH90], it is not easy to see whether FO^k suffices from the proof. Nevertheless, one can easily show that FO^4 suffices using Stockmeyer's result [Sto74], together with a reduction from [CH90], as we shall sketch next. We start with a result which is a simple corollary of Stockmeyer's well-known result [Sto74] that equivalence of star-free regular expressions over the alphabet $\{0, 1\}$ is nonelementary.

Proposition 5.5.3 *The FO^3 theory of the class \mathcal{C} of finite linear orders with a unary predicate is nonelementary.*

This proposition is a simple corollary of Stockmeyer's result [Sto74] and Proposition 2.5.1. To deduce Proposition 5.5.2, we may simply use the polynomial time reduction in [CH90] which, given a first-order sentence ϕ over \mathcal{C} , outputs a first-order sentence ψ with one extra variable such that ϕ is true in \mathcal{C} iff $(\langle \{0, 1\}^*, \text{succ}_0, \text{succ}_1, \preceq \rangle, \varepsilon) \models \psi$. By Proposition 5.5.3, the proof for Proposition 5.5.2 is complete.

Now define the transition system

$$\begin{aligned} \mathfrak{T} := & \langle \{0, 1\}^* \times \{0, 1\}^* \times \{0, 1\}^* \times \{0, 1\}^*; \\ & \{ \prec_0^i \}_{i=1}^4, \{ \prec_1^i \}_{i=1}^4, \{ <_i \}_{i=1}^4, \{ =_{i,j} \}_{1 \leq i < j \leq 4}, \{ G_i \}_{i=1}^4 \rangle. \end{aligned}$$

where the transition relations are defined as follows:

- $\prec_0^i := \{ (\overline{w}, \overline{w'}) : w'_i = w_i 0 \text{ and } \forall j \neq i (w_j = w'_j) \}$. This relation takes the i th component to its left child.
- $\prec_1^i := \{ (\overline{w}, \overline{w'}) : w'_i = w_i 1 \text{ and } \forall j \neq i (w_j = w'_j) \}$. This relation takes the i th component to its right child.
- $<_i := \{ (\overline{w}, \overline{w'}) : w_i \prec w'_i \text{ and } \forall j \neq i (w_j = w'_j) \}$. This relation takes the i th component to its descendant.
- $=_{i,j} := \{ (\overline{w}, \overline{w'}) : w_i = w_j \text{ and } \forall k (w_k = w'_k) \}$. This relation simply loops if the i th component equals the j th component.
- $G_i := \{ (\overline{w}, \overline{w'}) : \forall j \neq i (w_j = w'_j) \}$. This relation takes the i th component to any other word (i.e. global modality).

It is not difficult to give a word-automatic system that is isomorphic to \mathfrak{T} with a regular transitive closure relation (since we have the global modalities G_i).

Lemma 5.5.4 *The transition system \mathfrak{T} is automatically presentable with a regular transitive closure relation.*

Proof. Let $\Gamma := \{0, 1, \#\}$ and $\Sigma := \Gamma^4$. Given words $v_1, \dots, v_4 \in \{0, 1\}^*$, we define $v_1 \otimes' \dots \otimes v_4$ to be the word $v_1 \otimes \dots \otimes v_4$ but using $\#$ (instead of \perp) as the padding symbol, e.g., $0 \otimes' 11 \otimes' 1 \otimes' 101$ is simply

$$\begin{bmatrix} 0 \\ 1 \\ 1 \\ 1 \end{bmatrix} \begin{bmatrix} \# \\ 1 \\ \# \\ 0 \end{bmatrix} \begin{bmatrix} \# \\ \# \\ \# \\ 1 \end{bmatrix}.$$

Let $S := \{v_1 \otimes' \dots \otimes' v_4 : v_1, \dots, v_4 \in \Sigma^*\}$. We then define a relation R_a over S for each relation a in \mathfrak{T} in the obvious way. For example, the relation $R_{\prec_0^i}$ is defined as the relation $\{(v, v') \in S \times S : (v, v') \in \prec_0^i\}$. Let \mathfrak{T}' be the system with domain S and relations R_a , where a is a relation in \mathfrak{T} . It is easy to give an NWA \mathcal{A}_S over Σ recognizing the set S . Similarly, it is easy to construct an NWA \mathcal{A}_a over the alphabet $\Sigma_{\perp} \times \Sigma_{\perp}$ for the relations R_a . [These NWAs are very similar to the automata in Example 3.1.1.] Therefore, \mathfrak{T}' is a word-automatic system isomorphic to \mathfrak{T} . Finally, observe that the transitive closure of the transition relations in \mathfrak{T}' coincides with the regular relation $S \times S$, which completes the proof. \square

The following lemma is now sufficient to deduce Theorem 5.5.1.

Lemma 5.5.5 *Checking whether a given HM-logic formula ϕ over \mathfrak{T} is satisfied by $(\mathfrak{T}, (\varepsilon, \varepsilon, \varepsilon, \varepsilon))$ is nonelementary.*

Proof. We give a poly-time reduction from the FO^4 theory of $\langle \{0, 1\}^*, \text{succ}_0, \text{succ}_1, \preceq \rangle$. More precisely, we give a polynomial time computable function λ from FO^4 formulas $\phi(x_1, \dots, x_4)$ to HM-logic formulas $\lambda(\phi)$ over the vocabulary of \mathfrak{T} such that, for each $v_1, \dots, v_4 \in \{0, 1\}^*$,

$$\langle \{0, 1\}^*, \text{succ}_0, \text{succ}_1, \preceq \rangle \models \phi(v_1, \dots, v_4) \iff \mathfrak{T}, (v_1, \dots, v_4) \models \lambda(\phi) \quad (*)$$

The function λ is defined by induction on FO^4 formulas over $\langle \{0, 1\}^*, \text{succ}_0, \text{succ}_1, \preceq \rangle$. First consider the three base cases:

- We set $\lambda(x_i \prec_0^i x_j) := \langle \prec_0^i \rangle \langle =_{i,j} \rangle \top$.
- We set $\lambda(x_i \prec_1^i x_j) := \langle \prec_1^i \rangle \langle =_{i,j} \rangle \top$.
- We set $\lambda(x_i < x_j) := \langle <_{i,j} \rangle \langle =_{i,j} \rangle \top$.

It is easy to check that the statement $(*)$ hold for these. Now consider the inductive cases:

- We set $\lambda(\phi \wedge \phi') := \lambda(\phi) \wedge \lambda(\phi')$. It is easy to see that $(*)$ holds by inductive hypothesis.
- We set $\lambda(\neg \phi) := \neg \lambda(\phi)$. Clearly, the statement $(*)$ holds by inductive hypothesis.

- We set $\lambda(\exists x_i \varphi) := \langle G_i \rangle \lambda(\varphi)$. We now show that $(*)$ holds in this case. Without loss of generality, let $i = 1$ (the other cases are similar). Let $\psi := \exists x_1 \varphi$. Given $v_1, \dots, v_4 \in \{0, 1\}^*$, we have $\langle \{0, 1\}^*, \text{succ}_0, \text{succ}_1, \preceq \rangle \models \psi(v_1, \dots, v_4)$ iff there exists $v'_1 \in \{0, 1\}^*$ such that $\langle \{0, 1\}^*, \text{succ}_0, \text{succ}_1, \preceq \rangle \models \varphi(v'_1, v_2, v_3, v_4)$. By inductive hypothesis, the latter statement is true iff $\mathfrak{T}, (v'_1, v_2, v_3, v_4) \models \lambda(\varphi)$, which is true iff $\mathfrak{T}, (v_1, v_2, v_3, v_4) \models \langle G_i \rangle \lambda(\varphi)$ by the definition of G_i relation. This finishes the proof that $(*)$ holds in this case.

□

Chapter 6

More applications of algorithmic metatheorems

In this chapter, we will give other applications of our algorithmic metatheorems from the previous chapters. In particular, we will apply our algorithmic metatheorems to model checking problems over PA-processes, subclasses of Petri nets (e.g. reversible Petri nets and 2-dimensional vector addition systems), and reversal-bounded counter systems with discrete clocks and one free counter. There are other applications that we do not mention in this chapter, e.g., for deriving decidability of LTL with multi-regular fairness constraints over order-2 collapsible pushdown systems (combining with Kartzow’s recent result [Kar10]).

Notation. We define a notation for LTL model checking with multiple fairness constraints. Given an LTL formula ϕ over ACT, a transition system $\mathfrak{S} = \langle S, \{\rightarrow_a\}_{a \in \text{ACT}} \rangle$, and subsets $\{S_i\}_{i=1}^n$ of S , we write $[[\phi]]_{\mathfrak{S}, \{S_i\}_{i=1}^n}$ to denote the set of configurations $s \in S$ satisfying $(\mathfrak{S}, s, \{S_i\}_{i=1}^n) \models \phi$, i.e., for each infinite path $\pi = s_0 \rightarrow_{a_1} s_1 \rightarrow_{a_2} \dots$ in \mathfrak{S} satisfying $\exists^\infty i (s_i \in S_i)$ for each $j \in \mathbb{N}$, it is the case that $\pi \models \phi$. ■

6.1 PA-processes

PA [BW90, May98] is a well-known process algebra allowing sequential and parallel compositions, but no communication. It generalizes basic parallel processes (BPP), and context-free processes (BPA), but is incomparable to pushdown systems and Petri nets (e.g. see [May98]). PA has found applications in the interprocedural dataflow analysis of parallel programs [EP00].

We review the basic definitions, following the presentation of [LS02]: we initially distinguish terms that are equivalent up to simplification laws. Fix a finite set $Var = \{X, Y, Z, \dots\}$ of process variables. *Process terms* over Var , denoted by \mathcal{F}_{Var} , are generated by the grammar:

$$t, t' := 0 \mid X, X \in Var \mid t.t' \mid t \parallel t'$$

where 0 denotes a “nil” process, and $t.t'$ and $t \parallel t'$ are sequential and parallel compositions, respectively. Process terms can be viewed as Σ -labeled binary trees, where $\Sigma = Var \cup \{0, \parallel, \cdot\}$. In particular, inner nodes are always labeled by ‘ \cdot ’ or ‘ \parallel ’, while leaves are labeled by elements in $Var \cup \{0\}$. Observe that \mathcal{F}_{Var} is a regular tree language, for which a small NTA can be easily computed from any given Var . A PA declaration over ACT is a tuple $\mathcal{P} = (\text{ACT}, \mathcal{F}_{Var}, \Delta)$, where Δ is a finite of rewrite rules of the form (X, a, t) , where $X \in Var$, $a \in \text{ACT}$, and $t \in \mathcal{F}_{Var}$. We set $\text{Dom}(\Delta) = \{X : (X \rightarrow t) \in \Delta, \text{ for some } t \in \mathcal{F}_{Var}\}$, and $Var_0 = Var - \text{Dom}(\Delta)$. A PA declaration \mathcal{P} generates a transition relation $\mathfrak{S}_{\mathcal{P}} = \langle S, \{\rightarrow_a\}_{a \in \text{ACT}} \rangle$, where $S = \mathcal{F}_{Var}$ and \rightarrow_a is defined by the following inference rules:

$\frac{t_1 \rightarrow_a t'_1}{t_1 \parallel t_2 \rightarrow_a t'_1 \parallel t_2}$	$\frac{t_1 \rightarrow_a t'_1}{t_1.t_2 \rightarrow_a t'_1.t_2}$	$\overline{X \rightarrow_a t} \quad (X, a, t) \in \Delta$
$\frac{t_2 \rightarrow_a t'_2}{t_1 \parallel t_2 \rightarrow_a t_1 \parallel t'_2}$	$\frac{t_2 \rightarrow_a t'_2}{t_1.t_2 \rightarrow_a t_1.t'_2} \quad t_1 \in \text{IsNil}$	

Here IsNil is the set of “terminated” process terms, i.e., those in which all variables are in Var_0 . It is easy to give an NTA $\mathcal{R}_{\mathcal{A}}$ over $\text{TREE}_2(\Sigma_{\perp})$ for \rightarrow_a , whose size is linear in the size $\|\mathcal{P}\|$ of \mathcal{P} . It is defined in the same way as for GTRSs, except that when it guesses a leaf node v at which a rule is applied, it must further ensure that v has no ‘ \cdot ’-labeled ancestor u such that v is a descendant of u and that the subtree rooted at u is *not* a terminated process term. See [LS05a] for further details. The following proposition is a well-known result concerning PA.

Proposition 6.1.1 ([LS02, LS05a, EP00]) *Given a PA declaration Δ and a NTA \mathcal{A} describing a set of process terms over Var , the sets $\text{pre}^*(\mathcal{L}(\mathcal{A}))$ and $\text{post}^*(\mathcal{L}(\mathcal{A}))$ are regular, for which NTAs can be computed in time $O(|Var| \times \|\mathcal{P}\| \times \|\mathcal{A}\|)$, and one can construct an NTA \mathcal{R}^+ over $\text{TREE}_k(\Sigma_{\perp})$ for \rightarrow^+ in poly-time¹.*

In this section, we consider only tree languages that are interpreted as regular subsets of \mathcal{F}_{Var} . From Proposition 6.1.1, Theorem 5.2.4, and Theorem 5.2.7, the following

¹Lugiez and Schnoebelen first proved this in [LS02] for a notion of tree transducers, but later in [LS05a] realized that regular relations suffice

theorem is immediate.

Theorem 6.1.2 *Model checking LTL_{det} and $LTL(\mathbf{F}_s, \mathbf{G}_s)$ with multi-regular fairness constraints over PA are solvable in EXP and 2-EXP, respectively. For fixed formulas and fixed number of fairness constraints, these problems are solvable in P.*

In the study of PA processes, it is common to use a structural equivalence on process terms. We now extend our results to PA modulo structural equivalence. Let \equiv be the smallest equivalence relation on \mathcal{F}_{Var} that satisfies the following:

$$\begin{aligned} t.0 &\equiv t & 0.t &\equiv t & t\|0 &\equiv t & t\|t' &\equiv t'\|t \\ (t\|t')\|t'' &\equiv t\|(t'\|t'') & (t.t').t'' &\equiv t.(t'.t'') \end{aligned}$$

We let $[t]_{\equiv}$ stand for the equivalence class of t and $[L]_{\equiv}$ for $\bigcup_{t \in L} [t]_{\equiv}$. We write L/ \equiv for $\{[t]_{\equiv} : t \in L\}$. It was shown in [LS02] that, for each $t \in \mathcal{F}_{Var}$, $[t]_{\equiv}$ is a regular tree language, although the set $[L]_{\equiv}$ need not be regular even for regular L . Given a PA declaration \mathcal{P} and the transition system $\mathfrak{S}_{\mathcal{P}} = \langle S, \{\rightarrow_a\}_{a \in \text{ACT}} \rangle$ generated by \mathcal{P} , the equivalence \equiv generates a new transition system $(\mathfrak{S}_{\mathcal{P}}/ \equiv) = \langle S', \{\Rightarrow_a\}_{a \in \text{ACT}} \rangle$, where $S' = [S]/ \equiv$, and $[t]_{\equiv} \Rightarrow_a [u]_{\equiv}$ iff there exist $t' \in [t]_{\equiv}$ and $u' \in [u]_{\equiv}$ such that $t' \rightarrow_a u'$. We need the following result:

Lemma 6.1.3 ([LS02]) *The relation \equiv is bisimulation: for all $t, t', u \in \mathcal{F}_{Var}$, if $t \equiv t'$ and $t \rightarrow_a u$, then there exists $u' \in \mathcal{F}_{Var}$ such that $t' \rightarrow_a u'$ and $u \equiv u'$.*

Using this lemma, we may easily show that, for every sequence $\{\mathcal{A}_i\}_{i=1}^n$ of NTAs such that each $\mathcal{L}(\mathcal{A}_i)$ is closed under \equiv , and every LTL formula ϕ over ACT, the set $[[\phi]]_{\mathfrak{S}_{\mathcal{P}}, \{\mathcal{L}(\mathcal{A}_i)\}_{i=1}^n}$ is also closed under \equiv . This also implies that

$$[[\phi]]_{\mathfrak{S}_{\mathcal{P}}, \{\mathcal{L}(\mathcal{A}_i)\}_{i=1}^n} = [[[\phi]]_{\mathfrak{S}_{\mathcal{P}}, \{\mathcal{L}(\mathcal{A}_i)\}_{i=1}^n}]_{\equiv} = \{t : t \in [[\phi]]_{(\mathfrak{S}_{\mathcal{P}}/ \equiv), \{\mathcal{L}(\mathcal{A}_i)/ \equiv\}_{i=1}^n}\}.$$

The following theorem is then a direct consequence of Theorem 6.1.2.

Theorem 6.1.4 *Given a PA $\mathcal{P} = (\text{ACT}, \text{Var}, \Delta)$, a sequence of NTAs $\{\mathcal{A}_i\}_{i=1}^n$ such that each $\mathcal{L}(\mathcal{A}_i)$ is closed under \equiv , an LTL_{det} (resp. $LTL(\mathbf{F}_s, \mathbf{G}_s)$) formula ϕ over ACT, and a process term $t \in \mathcal{F}_{Var}$, it is possible to decide whether $(\mathfrak{S}_{\mathcal{P}}/ \equiv, [t]_{\equiv}, \{\mathcal{L}(\mathcal{A}_i)\}_{i=1}^n) \models \phi$ in time EXP (resp. 2-EXP). Furthermore, for a fixed formula and a fixed number of fairness constraints, the problem is solvable in P.*

To see this, since $[[\phi]]_{\mathfrak{S}_{\mathcal{P}}, \{\mathcal{L}(\mathcal{A}_i)\}_{i=1}^n} = [[[\phi]]_{\mathfrak{S}_{\mathcal{P}}, \{\mathcal{L}(\mathcal{A}_i)\}_{i=1}^n}]_{\equiv}$, we need only test whether $\mathfrak{S}_{\mathcal{P}}, t, \{\mathcal{L}(\mathcal{A}_i)\}_{i=1}^n \models \phi$, which can be done by appealing to Theorem 6.1.2.

The decidability result of Theorem 6.1.4 is known in the absence of multi-regular fairness constraints, but without complexity analysis [BKRS09, Reh07]. A natural question regarding Theorem 6.1.4 is whether multi-regular fairness constraints are useful since we need each constraint to be closed under \equiv . The answer is positive. Recall from Section 5.4 that multi-regular fairness constraints can be used to encode some natural fairness constraint considered in the verification of ground tree rewrite systems, e.g., that each leaf process will eventually be executed. The encoding of this constraint is by additionally coloring the leaf processes by the color ‘1’ or ‘2’. We then use two regular fairness constraints \mathcal{L}_1 and \mathcal{L}_2 , where \mathcal{L}_1 encodes the set of all trees all of whose leaves are colored 1, while \mathcal{L}_2 encodes the set of all trees all of whose leaves are colored 2. This encoding also works for PA since both \mathcal{L}_1 and \mathcal{L}_2 will be closed under \equiv .

6.2 Reversal-bounded counter systems

In this section, we combine our algorithmic metatheorems from the previous chapter with known results in the literature to obtain new results on model checking problems over reversal-bounded counter systems and their extensions with discrete clocks and one free counter, which were extensions of Ibarra’s reversal-bounded counter systems [Iba78] that were introduced in [DIB⁺00]. We first start with reversal-bounded counter systems with one free counter, and then extend the result when the systems have discrete clocks.

6.2.1 Basic model with one free counter

We first define the notion of reversal-bounded counter systems [Iba78]. First, the reader should review the definition of counter systems from Example 3.1.4. Consider now a k -counter system $\mathcal{M} = (\text{ACT}, X, Q, \Delta)$, the transition system $\mathfrak{S}_{\mathcal{M}}$ generated by \mathcal{M} , and a path

$$\pi := s_0 \rightarrow s_1 \rightarrow \dots \rightarrow s_m$$

in $\mathfrak{S}_{\mathcal{M}}$, where $s_i = (q_i, n_{1,i}, \dots, n_{k,i})$. Intuitively, the number of reversals on a counter j in π is defined to be the number of switches from a non-increasing mode to a non-decreasing mode (or vice versa) of the value of the counter j in π . For example, if the values of f_j are 1,1,1,2,3,4,3,2,2,3, then the number of reversals on the j th counter is 2. Let us make this definition more precise. For each $j \in [1, k]$, consider the function

$f_j : [0, m] \rightarrow \mathbb{N}$ defined by $f_j(i) = n_{j,i}$. Observe that f simply records the value of the counter j in configuration s_i . The *number of reversals on a counter j in π* is the number of indices $i \in [1, m-2]$ for which there exists $i' > i$ satisfying either

- (i) $n_{j,i+1} = n_{j,i} + 1$, (ii) $n_{j,i+1} = \dots = n_{j,i'}$, and (iii) $n_{j,i'+1} = n_{j,i'} - 1$, or
- (i) $n_{j,i+1} = n_{j,i} - 1$, (ii) $n_{j,i+1} = \dots = n_{j,i'}$, and (iii) $n_{j,i'+1} = n_{j,i'} + 1$.

We define the *number of reversals* in π to be the maximum number of reversal on all counters j in π . We say that \mathcal{M} is *r -reversal bounded* if the number of reversals of \mathcal{M} on all paths π in $\mathfrak{S}_{\mathcal{M}}$ is at most r . In addition, an r -reversal bounded counter system \mathcal{M} is said to have a *free counter* if there is no bound on the number of reversals made by the first counter.

We are interested in the problem of model checking LTL formulas over reversal-bounded k -counter systems $\mathcal{M} = (\text{ACT}, X, Q, \Delta)$, where $Q = \{q_1, \dots, q_n\}$, with multi-regular fairness constraints (again, see Example 3.1.4 for the representation of sets of configurations of \mathcal{M} using automata). Note that regular relations are more expressive than Presburger-definable relations as we have noted in Example 3.1.6.

Theorem 6.2.1 *Model checking an LTL formula ϕ with multi-regular fairness constraints over r -reversal bounded k -counter system with n states and one free counter can be done in time polynomial in the size of each fairness constraint, exponential in n and in the number of fairness constraints, but double exponential in r , k , and $\|\phi\|$.*

In the case of a single Presburger arithmetic fairness constraint, this problem was already known to be decidable [DIP01], but without any complexity analysis. In the next chapter, we shall lower the above upper bound complexity by one exponential when there is no extra free counter.

Before we apply our algorithmic metatheorems for proving Theorem 6.2.1, let us make a simple observation that the class of r -reversal bounded counter systems with one free counter (for any fixed $r \in \mathbb{N}$) is closed under products with finite systems.

Lemma 6.2.2 *Given an r -reversal bounded k -counter system $\mathcal{M} = (\text{ACT}, X, Q, \Delta)$ with one free counter and a finite system \mathfrak{F} over ACT, the product $\mathfrak{F} \times \mathfrak{S}_{\mathcal{M}}$ can be represented by an r -reversal bounded k -counter system \mathcal{M}' with one free counter, which can be computed in polynomial time.*

Let us now consider the reachability relations of reversal bounded counter systems with one free counter. We shall first discuss how we may represent the reachability relations

of a k -counter system \mathcal{M} with states $Q = \{q_1, \dots, q_n\}$ as subsets of \mathbb{N}^{2k} . For each pair $q_i, q_j \in Q$ of states, we may define the $2k$ -ary relation $R_{i,j}$ such that $(l_1, \dots, l_{2k}) \in R_{i,j}$ iff the configuration $(q_j, l_{k+1}, \dots, l_{2k})$ is reachable from (q_i, l_1, \dots, l_k) in $\mathfrak{S}_{\mathcal{M}}$. In other words, $R_{i,j}$ encodes the reachability relations of \mathcal{M} from configurations of the form (q_i, \mathbf{v}) to configurations of the form (q_j, \mathbf{w}) . In the sequel, if we say that the reachability relation of \mathcal{M} is semilinear, we mean that each $R_{i,j}$ is semilinear. We need the following result on the reachability relations of reversal-bounded counter systems.

Proposition 6.2.3 ([Iba78, ISD⁺02]) *The reachability relations of r -reversal bounded k -counter systems \mathcal{M} with one free counter are semilinear.*

Let us briefly discuss the proof ideas of this proposition. To do so, we must first define the notion of reversal-bounded counter automata, which can be understood as reversal-bounded counter systems used as language recognizers (the difference is similar to the difference between pushdown automata and pushdown systems). More precisely, an r -reversal bounded k -counter automaton over Σ is an r -reversal bounded counter system $\mathcal{M} = (\Sigma_{\epsilon}, X, Q, \delta)$, where $\Sigma_{\epsilon} = \Sigma \cup \{\epsilon\}$, with an initial state q_0 and a set F of final states. The language $\mathcal{L}(\mathcal{M})$ recognized by \mathcal{M} contains the set of words $w \in \Sigma^*$ for which there exists a path

$$(p_0, \mathbf{v}_0) \rightarrow_{a_1} \dots \rightarrow_{a_m} (p_m, \mathbf{v}_m)$$

in the transition system $\mathfrak{S}_{\mathcal{M}}$ generated by \mathcal{M} satisfying $p_0 = q_0$, $p_m \in F$, and $w = a_1 \dots a_m$. Such a path is said to be *accepting* and that w is said to be *accepted* by \mathcal{M} . Similar notions can be defined for reversal-bounded counter automata with one free counter. The first step in the proof is the following lemma from [ISD⁺02].

Lemma 6.2.4 ([ISD⁺02]) *There exists a polynomial-time algorithm which, given an r -reversal-bounded k -counter system with states $Q = \{q_1, \dots, q_n\}$ with (resp. without) one free counter, computes a r -reversal bounded k -counter automaton with (resp. without) one free counter such that the Parikh image of its language coincides with the reachability relations $\{R_{i,j}\}_{i,j \in [1,n]}$.*

The Parikh images of the language of r -reversal bounded k -counter automata with one free counter have been shown by Ibarra [Iba78] to be effectively semilinear, which immediately implies Proposition 6.2.3. We shall state Ibarra's result as a proposition since we will refer to it in the sequel.

Proposition 6.2.5 ([Iba78]) *Given a reversal-bounded counter automaton with one free counter, the Parikh image of its language is effectively semilinear.*

We now offer a slight modification of Ibarra's algorithm for computing the Parikh image of reversal-bounded counter automata to obtain an acceptable complexity upper bound. It is well-known that semilinear sets over \mathbb{N}^k coincide with subsets of \mathbb{N}^k that are definable in Presburger arithmetic [GS66]. If we represent semilinear sets as existential Presburger formulas with homogenous linear (in)equations of the form

$$a_1x_1 + \dots + a_mx_m \sim b,$$

where $a_1, \dots, a_m, b \in \mathbb{Z}$ and $\sim \in \{=, \neq, <, >, \leq, \geq\}$, as atomic formulas, it turns out that we can obtain the following upper bounds.

Proposition 6.2.6 *We can compute a representation of the Parikh image of the language of a given r -reversal bounded k -counter automaton \mathcal{M} with n states and one free counter as existential positive Presburger formulas with linear (in)equations (with unary representation of numbers) in time polynomial in n , and exponential in k and r .*

The complexity analysis of this proposition can simply be derived by replacing the use of the original proof of Parikh's Theorem [Par66] in Ibarra's proof of Proposition 6.2.5 by the polynomial-time algorithm from [VSS05] computing an existential positive Presburger formula with linear (in)equations which represents the Parikh image of a given context-free grammar (equivalently, pushdown automata).

Remark 6.2.1 In [Iba78], Ibarra did not provide complexity analysis of his algorithm. This complexity analysis, however, can be easily inferred by analyzing the algorithm from [Iba78] after using the algorithm from [VSS05] instead of [Par66]. ■

Using this proposition, it is immediate that the Parikh images of the language of reversal-bounded counter automata with one free counter are regular (in the sense given in Example 3.1.4) since linear (in)equations can always be replaced by Presburger formulas. In order to obtain compact automata representations, we will instead use the following translation from linear (in)equations to automata representations.

Proposition 6.2.7 ([BC96, WB00]) *Given a homogeneous linear (in)equation of the form*

$$a_1x_1 + \dots + a_mx_m \sim b$$

where $a_1, \dots, a_m, b \in \mathbb{Z}$, $\sim \in \{=, \neq, <, \leq, >, \geq\}$, the subset of \mathbb{N}^m containing valid valuations (i_1, \dots, i_m) of the variables (x_1, \dots, x_m) can be represented as an NWA \mathcal{A} over $\{0, 1\}^m$ (in the sense of Example 3.1.4) with $O(\sum_{i=1}^n |a_i| + \log(|b|))$ states. Furthermore, this can be computed in time polynomial in $\sum_{i=1}^n |a_i| + \log(|b|)$.

In other words, NWAs representing linear (in)equations can be represented polynomially in the size of the numbers (with unary representations), but exponentially in the number of summands. In fact, it is known that this upper bound cannot be substantially improved for DWAs [Kla08], which was recently shown to hold also for NWAs [DGH]. Proposition 6.2.7 can be combined with Proposition 3.1.3 from Chapter 3 to obtain a complexity bound on the size of NWA for existential positive Presburger formulas with linear (in)equations. The following proposition can be derived by combining Proposition 6.2.7, Proposition 3.1.3, Proposition 6.2.6, and Lemma 6.2.4.

Proposition 6.2.8 *There exists an algorithm which, given an r -reversal bounded k -counter system \mathcal{M} with n states and one free counter, computes an NWA \mathcal{A} representing the reachability relation of \mathcal{M} . Furthermore, the algorithm runs in time exponential in n but double exponential in r and k .*

Theorem 6.2.1 is then an immediate corollary of Proposition 6.2.8 and our algorithmic metatheorem for LTL model checking over word-automatic transition systems.

6.2.2 Extension with discrete clocks

We now give an extension of Theorem 6.2.1 with discrete clocks with no increase in computational complexity.

Let us first recall the definition of counter systems with discrete clocks [DIB⁺00]. An *atomic clock constraint* on clocks $Y = \{y_1, \dots, y_t\}$ is simply an expression of the form $y_i \sim y_j$ or $y_i - y_j \sim c$, where $\sim \in \{<, >, =\}$, $1 \leq i, j \leq t$ and $c \in \mathbb{Z}$ is a constant. Here c is given in *binary*. In the sequel, we shall call this constant c a *clock comparison constant*. An *atomic counter constraint* on counters $X = \{x_1, \dots, x_k\}$ is simply an expression of the form $x_i \sim 0$, where $\sim \in \{=, >\}$. A *counter-clock (CC) constraint* θ on (X, Y) is simply a conjunction of a clock constraint on Y and a counter constraint on X . Given a valuation $v : X \cup Y \rightarrow \mathbb{N}$ to the counter/clock variables, we can determine whether $\theta[v]$ is true or false in the obvious way. A *k -counter system with t discrete clocks over ACT* is a tuple $\mathcal{M} = (\text{ACT}, X, Y, Q, \Delta)$, where

- $X = \{x_1, \dots, x_k\}$ is a set of k counter variables,

- Q is a set of states,
- $Y = \{y_1, \dots, y_t\}$ is a set of t clock variables,
- Δ is a finite set of *instructions* of the form $((q, \theta(X, Y)), a, (q', \mathbf{v}, Y'))$, where:
 - $q, q' \in Q$,
 - $a \in \text{ACT}$,
 - $\theta(X, Y)$ is a CC constraint on (X, Y) ,
 - $\mathbf{v} \in \{-1, 0, 1\}^k$, and
 - $Y' \subseteq Y$ is a set of *clock resets*.

A *configuration* of \mathcal{M} is a tuple $(q, \mathbf{v}, \mathbf{w}) \in Q \times \mathbb{N}^k \times \mathbb{N}^t$ expressing the state \mathcal{M} is in, the current values of the k counter, and the current values of the t clocks. The k -counter system with t discrete clocks \mathcal{M} also generates a transition system $\mathfrak{S}_{\mathcal{M}} = \langle S, \{\rightarrow_a\}_{a \in \text{ACT}} \rangle$ defined as follows:

- $S = Q \times \mathbb{N}^k \times \mathbb{N}^t$ contains all configurations of \mathcal{M} .
- Given two configurations $\mathbf{c} = (q, \mathbf{v}, \mathbf{w})$ and $\mathbf{c}' = (q', \mathbf{v}', \mathbf{w}')$, we have $\mathbf{c} \rightarrow_a \mathbf{c}'$ iff there exists an instruction $((q, \theta(X, Y)), a, (q', \mathbf{u}, Y')) \in \Delta$ such that
 - $\theta[\mathbf{v}, \mathbf{w}]$ holds,
 - $\mathbf{v}' = \mathbf{v} + \mathbf{u}$, and
 - If $Y' = \emptyset$, then each clock progresses by one time unit: $\mathbf{w}' = \mathbf{w} + \mathbf{1}$. If $Y' \neq \emptyset$, then the value of the clocks in Y' are reset, while the values of other clocks stay the same: for each $y_i \in Y'$ and $y_j \in Y \setminus Y'$, $w'_i = 0$ and $w'_j = w_j$.

We may define the notions of reversal-bounded counter systems with one free counter and discrete clocks in the same way.

In order to apply our algorithmic metatheorems, first observe that the class of reversal-bounded counter systems with one free counter and discrete clocks are closed under products with finite systems. This observation is very similar to Lemma 6.2.2. We next consider their reachability relations. First recall the following proposition from [DIB⁺00].

Proposition 6.2.9 *Suppose that \mathcal{M} is an r -reversal bounded k -counter systems with t discrete clocks and with (resp. without) one free counter. Let n be the number of states*

of \mathcal{M} and l be the size (in binary) of the maximum absolute values of clock comparison constants in \mathcal{M} . Then, we may compute a $r + 1$ -reversal bounded $(k + t + 1)$ -counter automata \mathcal{M}' with (resp. without) one free counter such that $\mathcal{P}(\mathcal{L}(\mathcal{M}'))$ coincides with the reachability relation of \mathcal{M} . Furthermore, the procedure runs in time polynomial in n , k , and r , but exponential in l and t .

In fact, in the paper [DIP01] Dang *et al.* only gave the proof of this proposition when there is only one free counter and no reversal bounded counters, though they remarked that this can easily be extended with reversal bounded counters which can indeed be easily checked. Therefore, we may proceed as for the case without discrete clocks and obtain the following theorem.

Theorem 6.2.10 *Model checking an LTL formula ϕ with multi-regular fairness constraints over r -reversal bounded k -counter system \mathcal{M} with n states, t discrete clocks, and one free counter can be done in time polynomial in the size of each fairness constraint, exponential in n and in the number of fairness constraints, but double exponential in r , k , t , $\|\phi\|$, and the size (in binary) of the maximum absolute value of clock comparison constant in \mathcal{M} .*

This theorem also answers an open question by Dang *et al.* [DIP01] whether recurrent reachability with one Presburger-definable fairness constraint over reversal-bounded counter systems with discrete clocks and one free counter is decidable.

6.3 Subclasses of Petri nets

In this section, we shall apply our algorithmic metatheorems from earlier chapters to subclasses of Petri nets: (1) 2-dimensional vector addition systems with states, (2) reversible Petri nets, and (3) conflict-free Petri nets. Another subclass of Petri nets on which we can apply our algorithmic metatheorems is called basic parallel processes, which are a subclass of PA-processes which we considered earlier in this chapter.

Let us first recall the definition of vector addition systems with states. For our purpose, a *vector addition system with states* (VASS) is a tuple $\mathcal{P} = (\text{ACT}, X, Q, \delta)$, where

- $X = \{x_1, \dots, x_k\}$ is a set of *places*, and
- δ is a finite subset of $Q \times \text{ACT} \times \mathbb{Z}^k \times Q$ each of whose member is a *transition*.

The transition system $\mathfrak{S}_{\mathcal{P}} = \langle S, \{\rightarrow_a\}_{a \in \text{ACT}} \rangle$ generated by \mathcal{P} is defined as follows:

- $S = Q \times \mathbb{N}^k$, and
- for each action symbol $a \in \text{ACT}$, and pair of configurations $(q_1, \mathbf{v}_1), (q_2, \mathbf{v}_2) \in Q \times \mathbb{N}^k$, define $(q_1, \mathbf{v}_1) \rightarrow_a (q_2, \mathbf{v}_2)$ iff there exists a transition $(q_1, a, \mathbf{u}, q_2) \in \delta$ such that $\mathbf{w} = \mathbf{v} + \mathbf{u}$.

Observe that this definition enforces that no place could be made negative in any execution of the system. We define *Petri nets* to be vector addition systems with one state, in which case we will omit the state component when defining Petri nets. It is not hard to show that the transition systems generated by VASS and Petri nets are the same, i.e., for any given VASS \mathcal{P} , we can come up with one-state VASS that simulates \mathcal{P} . Finally, observe that VASS can be thought of as counter systems with no guard formulas.

6.3.1 Two-dimensional vector addition systems with states

Two-dimensional (2-dim) vector addition systems with states are simply VASS with two places. Leroux and Sutre [LS04] recently showed that the reachability relations of 2-dim VASS are effectively semilinear.

Proposition 6.3.1 ([LS04]) *The reachability relations of 2-dim VASS are effectively semilinear.*

This result generalizes an earlier result by Hopcroft and Pansiot [HP79] on the effective semilinearity of post^* and pre^* for 2-dim VASS. Nonetheless, no complexity analysis was provided in [LS04].

Observe now that VASS are word-automatic using the same encoding of counter systems as automata (see Example 3.1.4). Furthermore, we have remarked that semilinear sets (or equivalently Presburger-definable subsets of \mathbb{N}^k) can also be interpreted as regular languages using the same encoding of tuples of numbers. Since 2-dim VASS are closed under product with finite systems, our algorithmic metatheorem for decidable LTL model checking with multi-regular fairness constraints imply the following theorem.

Theorem 6.3.2 *Model checking LTL with multi-regular fairness constraints over 2-dim VASS is decidable.*

For this theorem, we could also replace regular fairness constraints with fairness constraints expressed as first-order formulas over Büchi Arithmetic (see Example 3.1.6), which generalizes Presburger Arithmetic. In addition, although it is known that model checking LTL over all VASS is EXPSPACE-complete [Yen92] (when only infinite paths are considered), to the best of our knowledge it is open whether the problem is decidable in the presence of semilinear (let alone, multi-regular) fairness constraints. At any rate, this problem is easily seen as hard as reachability for Petri nets, which is decidable but not known to be primitive recursive (cf. [May84]). As we remarked, the complexity of the construction of the reachability relations from [LS04] was not given. Therefore, we leave it as an open problem to pinpoint the precise complexity of this problem.

6.3.2 Conflict-free Petri nets

Let us briefly recall the definition of conflict-free Petri nets; for more details, the reader is referred to [Esp96]. Let $\mathcal{P} = (\text{ACT}, X, \delta)$ be a Petri net with k places. Given a configuration \mathbf{v} of \mathcal{P} , and a transition $t = (a, \mathbf{u})$ of \mathcal{P} , we say that t is *enabled at* \mathbf{v} if there exists $\mathbf{w} \in \mathbb{N}^k$ such that $\mathbf{w} = \mathbf{v} + \mathbf{u}$. In this case, we write $\mathbf{v} \xrightarrow{t} \mathbf{w}$ or simply $\mathbf{v} \xrightarrow{t}$ if \mathbf{w} is not important. We say that \mathcal{P} is *conflict free* if, for each $\mathbf{v} \in \mathbb{N}^k$, and pairs t_1, t_2 of transitions of \mathcal{P} , we have $\mathbf{v} \xrightarrow{t_1}$ and $\mathbf{v} \xrightarrow{t_2}$ implies $\mathbf{v} \xrightarrow{t_1} \mathbf{w} \xrightarrow{t_2}$ for some $\mathbf{w} \in \mathbb{N}^k$. The reachability relations of conflict-free Petri nets are known to be effectively semilinear.

Proposition 6.3.3 ([LS05a]) *The reachability relations of conflict-free Petri nets are effectively semilinear.*

Combining this with our algorithmic metatheorem for recurrent reachability with generalized Büchi conditions, we obtain the following theorem.

Theorem 6.3.4 *Checking recurrent reachability with multi-regular fairness constraints over conflict-free Petri nets is decidable.*

Since no complexity analysis was provided in [LS05a], we leave the complexity of this problem for future work. Finally, since it is easy to see that conflict free Petri nets are closed under taking subsystems, our algorithmic metatheorems for LTL_{det} and $\text{LTL}(\mathbf{F}_s, \mathbf{G}_s)$ imply the following theorem.

Theorem 6.3.5 *Model checking LTL_{det} and $\text{LTL}(\mathbf{F}_s, \mathbf{G}_s)$ with multi-regular fairness constraints over conflict-free Petri nets is decidable.*

6.3.3 Reversible Petri nets

A Petri net $\mathcal{P} = (\text{ACT}, X, \delta)$ is said to be *reversible* if for each transition $t \in \delta$, there exists a transition $t' \in \delta$ such that \xrightarrow{t} is an inverse of the relation $\xrightarrow{t'}$, i.e.,

$$\xrightarrow{t} = \{(\mathbf{v}, \mathbf{w}) : \mathbf{w} \xrightarrow{t'} \mathbf{v}\}.$$

For a more thorough treatment of reversible Petri nets, we refer the reader to [Esp96]. The reachability relations of reversible Petri nets are known to be effectively semilinear.

Proposition 6.3.6 ([LS05a]) *The reachability relations of reversible Petri nets are effectively semilinear.*

Combining this with our algorithmic metatheorem for recurrent reachability with generalized Büchi conditions, we obtain the following theorem.

Theorem 6.3.7 *Checking recurrent reachability with multi-regular fairness constraints over reversible Petri nets is decidable.*

Again, since no complexity analysis was provided in [LS05a], we leave the complexity of this problem for future work.

Part II

Specific Approaches

Chapter 7

Reversal-bounded counter systems and their extensions

Minsky's counter systems are well-known Turing-powerful models of computation. Hence, to obtain decidability, restrictions need to be imposed. We have seen in Chapter 6 that reversal-bounded counter systems, which were initially proposed by Ibarra [Iba78], are not Turing-powerful since they have decidable reachability and LTL model checking with complex fairness constraints. Furthermore, we saw that this is true even in the presence of one free counter and any number of discrete clocks. In this chapter, we shall investigate such models more thoroughly, especially with regards to the precise complexity of model checking problems.

We saw in Chapter 6 that our algorithmic metatheorem for LTL model checking with fairness constraints, combined with the result of [DIB⁺00, ISD⁺02, Iba78], yields decidability of LTL with fairness constraints over r -reversal k -counter systems with t discrete clocks and one free counter, which was left open by Dang *et al.* [DIP01]. In fact, the complexity upper bound that we obtained was double exponential time, even for a fixed LTL formula. This is far worse than the best known complexity lower bound for the problem, which is PSPACE-hard due to the presence of an unbounded number of clocks or binary representation of numbers in the clock constraints [CY92] (also see [AM04]). In this chapter, we shall rectify this problem in the case of reversal-bounded counter systems with discrete clocks, but without one free counter.

Recall from Chapter 6 that the results of [DIB⁺00, ISD⁺02, Iba78] yield a double-exponential time procedure for computing an NWA representing the reachability relation of a given r -reversal k -counter systems \mathcal{P} with t discrete clocks and one free counter. More precisely, if n is the number of states of \mathcal{P} , the complexity of this pro-

cedure is exponential in n , and double exponential in r (in unary), k , t , and the size of the binary representation of the maximum number appearing in clock constraints in \mathcal{P} . We observe that the constructions from [DIB⁺00, ISD⁺02, Iba78] substantially rely on Ibarra's original algorithm [Iba78] for the computation of semilinear sets representing the Parikh image of the language recognized by reversal-bounded counter machines.

Ibarra's original algorithm [Iba78] has been observed by Gurari and Ibarra [GI81] to give non-optimal algorithms for solving various problems (e.g. nonemptiness) for reversal-bounded counter machines. Gurari and Ibarra proposed a new technique for deriving a PSPACE upper bound for nonemptiness (and therefore reachability) for reversal-bounded counter machines. In fact, the procedure of [GI81] runs in polynomial-time when the parameters r and k are fixed constants. Later, Howell and Rosier [HR87] improved both the upper bound and lower bound for nonemptiness of reversal-bounded counter machines even in presence of one free counter. They showed that the problem is NP-complete when at least one of the parameters r and k is *not* fixed. Again, the technique of Howell and Rosier's avoids the use of Ibarra's original algorithm from [Iba78].

A careful look at Ibarra's algorithm [Iba78] reveals that the bottleneck of its running time is due to the use of Parikh's Theorem [Par66]. It can be easily checked that Parikh's construction of the Parikh images for CFGs (or, equivalently pushdown automata) runs in exponential time and may output a union of *exponentially* many linear sets in the worst case. In the case of r -reversal k -counter machines without one free counter, Parikh's construction was applied in [Iba78] to an NWA that is obtained from the input reversal-bounded counter machine of size exponential in r and k , and polynomial in the number n of states of the counter machine. The exponentiability of Parikh's construction then gives double exponential complexity in r and k , and exponential in n for computing the Parikh images of reversal-bounded counter machines. Although several different proofs for Parikh's Theorem with different flavours and techniques exist in the literature (e.g. see [Esp97b, Koz97, SSMH04, VSS05]), it can be easily checked that all of these constructions could produce at least exponentially many linear sets in the worst case *even when the size of the alphabet is restricted to one*.

The first hint that better upper bounds could be obtained for NWAs is due to Chrobak [Chr86] and Martinez [Mar02], who showed that there exists a polynomial-time algorithm which, given an NWA \mathcal{A} over an alphabet Σ of size 1, computes a union $\bigcup_{i=1}^m \{a_i + tb_i : t \in \mathbb{N}\}$ of polynomially many arithmetic progressions — whose offsets a_i and periods b_i are bounded polynomially in the number of states — representing

the Parikh image $\mathcal{P}(\mathcal{A})$ of \mathcal{A} . [Unfortunately, the proofs in [Chr86, Mar02] contain a subtle error, which were recently fixed by the author in [To09b].] A generalization of Chrobak-Martinez’s Theorem has been recently discovered independently by Kopczynski [Kop10] and the author [To10] with different proofs (see also the merged paper [KT10]). These results give an algorithm which, given an NWA with n states over an alphabet of size $k \geq 1$, computes a union of linear sets with at most k periods and total size $2^{O(k^2 \log n)}$ (with unary representation of numbers in the output), i.e., polynomial for all fixed k . Previously, it was not even known whether the number of periods in the linear sets could be made independent of n except for the special cases when $k = 1$ [Chr86] and $k = 2$ [Abe95]. Furthermore, the running time of our algorithms is polynomial in n and exponential in k , i.e., polynomial when k is fixed. This chapter primarily aims to present the author’s proof [To10] of this *normal form theorem for NWAs*, and show how they can be applied to obtain optimal model checking complexities of reversal-bounded counter systems and their extensions with discrete clocks.

This chapter is organized as follows. In Section 7.2, we prove a “Caratheodory-like” theorem for linear sets. *Caratheodory’s Theorem for convex cones* is a well-known result from the study of convex sets [Zie07] that over \mathbb{R}^k the *convex cone* generated by the vectors in $V = \{\mathbf{v}_1, \dots, \mathbf{v}_m\} \subseteq \mathbb{R}^k$ could be subdivided into convex cones that are generated by subsets $S \subseteq V$ of size at most k . More precisely, this fact can be written as

$$\mathbf{cone}(V) = \bigcup_{S \subseteq V, |S| \leq k} \mathbf{cone}(S),$$

where, if $S = \{\mathbf{w}_1, \dots, \mathbf{w}_r\}$, we define $\mathbf{cone}(S) := \{\sum_{i=1}^r t_i \mathbf{w}_i : t_1, \dots, t_r \in \mathbb{R}_{\geq 0}\}$. Observe that the number of cones on the right is only exponential in k (and polynomial in m). Our Caratheodory-like theorem for linear sets simply says that, given the set $V = \{\mathbf{v}_1, \dots, \mathbf{v}_m\} \subseteq \mathbb{N}^k$, the linear set $P(\mathbf{0}; V)$ can be written as a union of r linear sets $P(\mathbf{w}_1; S_1), \dots, P(\mathbf{w}_r; S_r)$, where each S_i is a subset of V of size at most k and the parameter r , as well as each number in each offset \mathbf{w}_i , is bounded exponentially in the dimension k , but polynomially in m and the maximum number a occurring in vectors in V . Furthermore, the linear sets $P(\mathbf{w}_1; S_1), \dots, P(\mathbf{w}_r; S_r)$ can be computed in polynomial time provided that the dimension k is fixed. In fact, we shall prove a more general version of this Caratheodory-like for \mathbb{Z} -linear sets. In Section 7.3, we shall use this Caratheodory-like theorem for linear sets to obtain a normal form theorem for Parikh images of NWAs. We shall also show in this section that the up-

per bound complexity given by this normal form theorem is tight and that the same upper bound does *not* hold for CFGs even over the fixed alphabet $\Sigma = \{a\}$. Before showing how the normal form theorem can be used to improve model checking complexities of reversal-bounded counter systems and their extensions, in Section 7.4 we give three simple applications of our Caratheodory-like theorem for linear sets and normal form theorem for Parikh images of NWAs: (1) polynomial-time fragments of integer linear programming, (2) decision problems for Parikh images of NWAs, and (3) Presburger-constrained graph reachability. In Section 7.5, we use our normal form theorem to obtain optimal complexity for the problem of model checking LTL with complex fairness constraints and EF-logic over reversal-bounded counter systems and their extensions with discrete clocks. Moreover, we shall show that model checking CTL is undecidable over reversal-bounded counter systems *without* discrete clocks.

7.1 Preliminaries

In this section, we shall define notations that will be used throughout this chapter. Fix an NWA \mathcal{A} over some alphabet Σ . Given a path $\pi = p_0 a_1 \dots p_m$ in an NWA \mathcal{A} , we shall write $\mathcal{P}(\pi)$ to denote the Parikh image $\mathcal{P}(a_1 \dots a_m)$ of the path labels $a_1 \dots a_m$.

We shall now fix some matrix notations. Given two n -by- n 0-1 matrices $M = [m_{i,j}]_{n \times n}$ and $M' = [m'_{i,j}]_{n \times n}$, we write $M \bullet M'$ to denote the matrix $M'' = [m''_{i,j}]_{n \times n}$ with $m''_{i,j} = \bigvee_{k=1}^n (m_{i,k} \wedge m'_{k,j})$. The operator \bullet is often referred to as *boolean matrix multiplication*, which can easily be evaluated in $O(n^3)$. We also write $M \vee M'$ to denote the application of the boolean operation \vee component-wise, i.e., resulting in a matrix $M'' = [m''_{i,j}]_{n \times n}$ with $m''_{i,j} = m_{i,j} \vee m'_{i,j}$. In the sequel, we shall also write $M[i, j]$ for the (i, j) -component $m_{i,j}$ of M .

Above we have defined the notions of convex cones. We now define a similar notion when the coefficients of the linear combinations are naturals (instead of non-negative reals). Given a finite subset S of vectors over \mathbb{Z}^k , let $\mathbf{cone}_{\mathbb{N}}(S)$ denote the linear set $P(\mathbf{0}; S)$. We now state a very simple fact about arithmetic on semilinear sets which we will frequently use in this chapter.

Fact 7.1.1 *Suppose that $S_1 = \bigcup_{i=1}^r P(\mathbf{v}_i; V_i) \subseteq \mathbb{N}^k$ and $S_2 = \bigcup_{j=1}^l P(\mathbf{w}_j; W_j) \subseteq \mathbb{N}^k$. Then, it is the case that $S_1 + S_2 = \bigcup_{i=1}^r \bigcup_{j=1}^l P(\mathbf{v}_i + \mathbf{w}_j; V_i \cup W_j)$. In addition, we have $P(\mathbf{v}; S) = \mathbf{v} + \mathbf{cone}_{\mathbb{N}}(S)$.*

7.2 A Caratheodory-like theorem for linear sets

In this section, we shall prove a Caratheodory-like theorem for linear sets of the form $\mathbf{cone}_{\mathbb{N}}(V)$. This will also imply a Caratheodory-like theorem for general linear sets $P(\mathbf{v}; V) = \mathbf{v} + \mathbf{cone}_{\mathbb{N}}(V)$.

Theorem 7.2.1 *Let $V := \{\mathbf{v}_1, \dots, \mathbf{v}_m\} \subseteq \mathbb{Z}^k \setminus \{\mathbf{0}\}$ with $m > 0$. Let $a \in \mathbb{N}$ be the maximum absolute value of numbers appearing in vectors of V . Then, it is possible to compute in time $2^{O(k \log(m) + k^2 \log(ka))}$ a sequence of \mathbb{Z} -linear bases $\langle \mathbf{w}_1; S_1 \rangle, \dots, \langle \mathbf{w}_\mu; S_\mu \rangle$ such that*

$$\mathbf{cone}_{\mathbb{N}}(V) = \bigcup_{i=1}^{\mu} P(\mathbf{w}_i; S_i)$$

where the maximum absolute value of entries of each \mathbf{w}_i is $O(m(k^2 a)^{2k+3})$, each S_i is a subset of V with $|S_i| \leq k$, and $\mu = O(m^{2k}(k^2 a)^{2k+3k})$. Furthermore, if $V \subseteq \mathbb{N}^k$, we have $\{\mathbf{w}_1, \dots, \mathbf{w}_\mu\} \subseteq \mathbb{N}^k$.

Observe that this theorem causes only an exponential blow-up in the dimension k . Moreover, each set S_i contains at most k generators. To prove this theorem, we start with a slight strengthening of *the conical version of Caratheodory's theorem* from the theory of convex sets [Zie07, Proposition 1.15]. The proof is given in the appendix.

Lemma 7.2.2 *Let $V := \{\mathbf{v}_1, \dots, \mathbf{v}_m\} \subseteq \mathbb{Z}^k \setminus \{\mathbf{0}\}$ with $m > 0$. Let $a \in \mathbb{N}$ be the maximum absolute value of numbers appearing in vectors of V . Then, it is possible to compute in time $2^{O(k \log m + \log \log a)}$, a sequence S_1, \dots, S_r of distinct linearly independent subsets of V with d elements, where $d \in \{1, \dots, k\}$ is the rank of V , and*

$$\mathbf{cone}(V) = \bigcup_{i=1}^r \mathbf{cone}(S_i).$$

Let us first explain the idea behind the rest of the proof of Theorem 7.2.1. Intuitively, Lemma 7.2.2 says that $\mathbf{cone}(V) \subseteq \mathbb{R}^k$ can be subdivided into smaller subcones with exactly $d \in \{1, \dots, k\}$ generators where d is the rank of V . This lemma immediately gives an *upper bound* for $\mathbf{cone}_{\mathbb{N}}(V)$ as the union of the *integer points* in $\mathbf{cone}(S_i)$; in general, the latter contains many more points than $\mathbf{cone}_{\mathbb{N}}(V)$. On the other hand, we have $\bigcup_{i=1}^r \mathbf{cone}_{\mathbb{N}}(S_i) \subseteq \mathbf{cone}_{\mathbb{N}}(V)$, where the inclusion is strict in general. It turns out that an equality can be achieved by first making a “few” *duplicates* of each $\mathbf{cone}_{\mathbb{N}}(S_i)$ and then *shifting* them appropriately by some “small” integer vectors.

We now prove Theorem 7.2.1. First invoke Lemma 7.2.2 on V and obtain linearly independent d -subsets S_1, \dots, S_r of V , where $d = \mathbf{rank}(V)$ and $r \leq m^k$, satisfying $\mathbf{cone}(V) = \bigcup_{j=1}^r \mathbf{cone}(S_j)$. Then, it follows that $\mathbf{cone}(V) \cap \mathbb{Z}^k = \bigcup_{j=1}^r (\mathbf{cone}(S_j) \cap \mathbb{Z}^k)$. To compute the integer vector “shifts”, we shall need to define the notions of *canonical* and *minimal* vectors.

Characterization via canonical and minimal vectors

Suppose now that $\mathbf{v} \in \mathbf{cone}(S_j) \cap \mathbb{Z}^k$ and $S_j = \{\mathbf{u}_1, \dots, \mathbf{u}_d\}$. We make several simple observations:

- (O1) There exists a *unique* vector $[\mathbf{v}] \in \{-ka, \dots, ka\}^k \cap \mathbf{cone}(S_j)$ and *unique* non-negative integers s_1, \dots, s_d such that: 1) $\mathbf{v} = [\mathbf{v}] + \sum_{i=1}^d s_i \mathbf{u}_i$, and 2) $[\mathbf{v}] = \sum_{i=1}^d t_i \mathbf{u}_i$ for some (unique) $0 \leq t_1, \dots, t_d < 1$. To see this, observe that by linear independence of S_j there exist some unique $\lambda_1, \dots, \lambda_d \in \mathbb{R}_{\geq 0}$ such that $\mathbf{v} = \sum_{i=1}^d \lambda_i \mathbf{u}_i$. Simply let $s_i := \lfloor \lambda_i \rfloor$, $t_i := \lambda_i - s_i$, and $[\mathbf{v}] := \sum_{i=1}^d t_i \mathbf{u}_i$. Uniqueness is immediate from uniqueness of $\lambda_1, \dots, \lambda_d$.
- (O2) Given $\mathbf{v}' \in \mathbf{cone}(S_j) \cap \mathbb{Z}^k$, we write $\mathbf{v} \sim \mathbf{v}'$ iff $[\mathbf{v}] = [\mathbf{v}']$. It is easy to see that \sim is an equivalence relation of finite index (there are at most $(2ka + 1)^k$ equivalence classes). If $[\mathbf{v}] = \mathbf{v}$, the vector \mathbf{v} is said to be a *canonical representative* of the equivalence class $\{\mathbf{u} \in \mathbf{cone}(S_j) \cap \mathbb{Z}^k : [\mathbf{u}] = \mathbf{v}\}$. In this case, we will also call \mathbf{v} an *S_j -canonical vector*, or simply *canonical vector* when S_j is understood.
- (O3) If \mathbf{v} is in $\mathbf{cone}_{\mathbb{N}}(V) \cap \mathbf{cone}(S_j)$, then $\mathbf{v} + \sum_{i=1}^d s_i \mathbf{u}_i \in \mathbf{cone}_{\mathbb{N}}(V) \cap \mathbf{cone}(S_j)$ for every $s_1, \dots, s_d \in \mathbb{N}$.

We shall now use these observations to define a natural well-founded partial order \leq_j on $\mathbf{cone}_{\mathbb{N}}(V) \cap \mathbf{cone}(S_j)$; note that $\mathbf{cone}_{\mathbb{N}}(V) \cap \mathbf{cone}(S_j) \neq \emptyset$. Given $\mathbf{v}, \mathbf{w} \in \mathbf{cone}_{\mathbb{N}}(V) \cap \mathbf{cone}(S_j)$, we write $\mathbf{v} \leq_j \mathbf{w}$ iff, for some (unique) S_j -canonical vector \mathbf{v}_0 and some (unique) coefficients $s_1, \dots, s_d \in \mathbb{N}$ and $t_1, \dots, t_d \in \mathbb{N}$, it is the case that: 1) $\mathbf{v} = \mathbf{v}_0 + \sum_{i=1}^d s_i \mathbf{u}_i$, 2) $\mathbf{w} = \mathbf{v}_0 + \sum_{i=1}^d t_i \mathbf{u}_i$, and 3) $(s_1, \dots, s_d) \preceq (t_1, \dots, t_d)$. The following simple lemma shows that \leq_j is a well-founded partial order, and characterizes \leq_j -minimal elements.

Lemma 7.2.3 *The relation \leq_j is a well-founded partial order on $\mathbf{cone}_{\mathbb{N}}(V) \cap \mathbf{cone}(S_j)$. Furthermore, a vector $\mathbf{v} \in \mathbf{cone}_{\mathbb{N}}(V) \cap \mathbf{cone}(S_j)$ is \leq_j -minimal iff none of the vectors $(\mathbf{v} - \mathbf{u}_1), \dots, (\mathbf{v} - \mathbf{u}_d)$ are in $\mathbf{cone}_{\mathbb{N}}(V) \cap \mathbf{cone}(S_j)$.*

Proof. That \leq_j is a partial order is due to:

- Observations **(O1)** and **(O2)**: the uniqueness of choice of canonical vector \mathbf{v}_0 and coefficients $s_1, \dots, s_d \in \mathbb{N}$ for each vector \mathbf{v} satisfying $\mathbf{v} = \mathbf{v}_0 + \sum_{i=1}^d s_i \mathbf{u}_i$.
- That \preceq is a partial order on \mathbb{N}^k .

To see that \leq_j is well-founded, assume that there exists a strictly decreasing sequence $\mathbf{v}_1 \triangleright_j \mathbf{v}_2 \triangleright_j \dots$. Let $\mathbf{v}_i = \mathbf{v}_0 + \sum_{j=1}^d s_j^i \mathbf{u}_j$ for some unique canonical vector $\mathbf{v}_0 = [\mathbf{v}_i]$ and unique coefficients $\mathbf{a}^i = (s_1^i, \dots, s_d^i) \in \mathbb{N}^d$. In this way, we generate a strictly decreasing sequence $\mathbf{a}^1 \succ \mathbf{a}^2 \succ \dots$ for the well-founded partial order \succ on \mathbb{N}^k , and therefore a contradiction. Thus, \leq_j is a well-founded partial order on $\mathbf{cone}_{\mathbb{N}}(V) \cap \mathbf{cone}(S_j)$.

Given a \leq_j -minimal vector $\mathbf{v} \in \mathbf{cone}_{\mathbb{N}}(V) \cap \mathbf{cone}(S_j)$, it is obvious that none of the vectors $(\mathbf{v} - \mathbf{u}_1), \dots, (\mathbf{v} - \mathbf{u}_d)$ cannot be in $\mathbf{cone}_{\mathbb{N}}(V) \cap \mathbf{cone}(S_j)$. Conversely, given a vector $\mathbf{v} \in \mathbf{cone}_{\mathbb{N}}(V) \cap \mathbf{cone}(S_j)$ which is not \leq_j -minimal, we could find another vector $\mathbf{v}' \in \mathbf{cone}_{\mathbb{N}}(V) \cap \mathbf{cone}(S_j)$ such that $\mathbf{v}' \triangleleft_j \mathbf{v}$. Using Observation **(O3)**, it is easy to show that at least one of the vectors $(\mathbf{v} - \mathbf{u}_1), \dots, (\mathbf{v} - \mathbf{u}_d)$ is in $\mathbf{cone}_{\mathbb{N}}(V) \cap \mathbf{cone}(S_j)$. \square

Lemma 7.2.3 and Observation **(O3)** immediately implies that $\mathbf{cone}_{\mathbb{N}}(V)$ is a union of linear sets $P(\mathbf{v}; S_j)$ taken over all $j = 1, \dots, r$ and \leq_j -minimal vectors \mathbf{v} .

Lemma 7.2.4 *The following equality holds*

$$\mathbf{cone}_{\mathbb{N}}(V) = \bigcup_{j=1}^r \bigcup_{\mathbf{v}} P(\mathbf{v}; S_j),$$

where \mathbf{v} is taken over all \leq_j -minimal vectors.

Proof. (\supseteq) Obvious.

(\subseteq) If $\mathbf{v} \in \mathbf{cone}_{\mathbb{N}}(V)$, then $\mathbf{v} \in \mathbf{cone}(S_j) \cap \mathbb{Z}^k$ for some $j \in \{1, \dots, r\}$. By Lemma 7.2.3, there exists a \leq_j -minimal vector \mathbf{v}' satisfying $\mathbf{v}' \leq_j \mathbf{v}$. Observation **(O3)** implies that $\mathbf{v} \in P(\mathbf{v}'; S_j)$. \square

Note also that if $V \subseteq \mathbb{N}^k$, then all \leq_j -minimal vectors ($1 \leq j \leq r$) are also nonnegative.

A roadmap for rest of the proof is as follows. We shall show that each \leq_j -minimal vectors cannot be too large and can be efficiently enumerated. This will immediately give us the desired sequence of linear bases. The proof of this will require connections to integer programming, and the use of dynamic programming.

Bounds via integer programming

For each $S_j = \{\mathbf{u}_1, \dots, \mathbf{u}_d\}$, we shall now show that all \preceq_j -minimal vectors \mathbf{v} cannot be too large. To this end, for each canonical vector $\mathbf{v}_0 \in \mathbf{cone}(S_j) \cap \mathbb{Z}^k$, consider the integer linear program $\mathbf{A}\mathbf{x} = \mathbf{v}_0^T$ ($\mathbf{x} \succeq \mathbf{0}$), where A is the $k \times (m+d)$ matrix consisting of columns $\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_m, -\mathbf{u}_1, -\mathbf{u}_2, \dots, -\mathbf{u}_d$ (in this order) and \mathbf{x} is the column $(m+d)$ -vector consisting of the variables $x_1, \dots, x_m, y_1, \dots, y_d$ (in this order). The following simple lemma shows that \preceq -minimal solutions to such integer programs — as we shall see, they cannot be too large as well — provide upper bounds for how large \preceq_j -minimal vectors can be.

Lemma 7.2.5 *For every \preceq_j -minimal vector $\mathbf{v} \in \mathbf{cone}_{\mathbb{N}}(V) \cap \mathbf{cone}(S_j)$, let $\mathbf{v}_0 := \lfloor \mathbf{v} \rfloor$ and $\mathbf{t} = (t_1, \dots, t_d) \in \mathbb{N}^d$ be the unique coefficients such that $\mathbf{v} = \mathbf{v}_0 + \sum_{i=1}^d t_i \mathbf{u}_i$. Suppose also that $\mathbf{c} = (c_1, \dots, c_m)$ is a \preceq -minimal solution to the integer program $\sum_{i=1}^m x_i \mathbf{v}_i = \mathbf{v}$ ($\mathbf{x} \succeq \mathbf{0}$). Then, the vector $\mathbf{w} := (\mathbf{c}, \mathbf{t}) \in \mathbb{N}^{m+d}$ is a \preceq -minimal solution to the integer program $\mathbf{A}\mathbf{x} = \mathbf{v}_0$ ($\mathbf{x} \succeq \mathbf{0}$).*

Proof. That \mathbf{w} is a solution is immediate. To show \preceq -minimality, consider a vector $\mathbf{u} = (c'_1, \dots, c'_m, t'_1, \dots, t'_d) \in \mathbb{N}^{m+d}$ such that $\mathbf{u} \preceq \mathbf{w}$ and $\mathbf{A}\mathbf{u}^T = \mathbf{v}_0$. Define $\mathbf{v}' := \mathbf{v}_0 + \sum_{i=1}^d t'_i \mathbf{u}_i$ and thus $\mathbf{v}' = \sum_{i=1}^m c'_i \mathbf{v}_i$. This means that $\mathbf{v}' \in \mathbf{cone}_{\mathbb{N}}(V) \cap \mathbf{cone}(S_j)$ and, by \preceq_j -minimality of \mathbf{v} , it follows that $\mathbf{v}' = \mathbf{v}$ and thus $t'_i = t_i$ for every $1 \leq i \leq d$. That \mathbf{c} is a \preceq -minimal solution to the integer program $\sum_{i=1}^m x_i \mathbf{v}_i = \mathbf{v}$ ($\mathbf{x} \succeq \mathbf{0}$) implies that $c'_i = c_i$ for every $1 \leq i \leq m$ and, thus, $\mathbf{u} = \mathbf{w}$. \square

Consider the set U of all vectors $\mathbf{v}_0 + \sum_{i=1}^d s_i \mathbf{u}_i$, where \mathbf{v}_0 ranges over all canonical vectors and s_1, \dots, s_d ranges over all d -tuples of nonnegative integers such that $(c_1, \dots, c_m, s_1, \dots, s_d)$ is a \preceq -minimal solution to the integer program $\mathbf{A}\mathbf{x} = \mathbf{v}_0$, for some $c_1, \dots, c_m \in \mathbb{N}$. We shall see now that the maximum absolute value B of numbers appearing in U exists, which immediately gives an upper bound for the maximum absolute value of entries of \preceq_j -minimal vectors. The following general lemma, whose proof is a straightforward adaptation of the proof of [Pap81, Theorem p. 767], yields an upper bound for B .

Lemma 7.2.6 *Let A be a $k \times n$ integer matrix and \mathbf{b} a k -vector, both with entries in $[-t, t] \cap \mathbb{Z}$, where $t \in \mathbb{N}$. Then, every \preceq -minimal solution $\mathbf{x} \in \mathbb{N}^n$ to $\mathbf{A}\mathbf{x} = \mathbf{b}$ ($\mathbf{x} \succeq \mathbf{0}$) is in $\{0, 1, \dots, n(kt)^{2k+1}\}^n$.*

Notice that the maximum absolute value of numbers appearing in our integer programs cannot exceed $t := ak$ (which could appear on the right hand side of the equation). If $M := (m+k)(kt)^{2k+1}$, it follows that $B \leq akM + ak \leq N := (m+k)(k^2a)^{2k+2} + ak$. This completes the proof of *existence* for Theorem 7.2.1 and gives us the desired bounds for the parameter μ and the maximum absolute value of entries of each w_i in Theorem 7.2.1. It remains to show how to make this algorithmic.

Computing canonical and minimal vectors

We first show how to compute all the canonical vectors. Since Gaussian-elimination over rational numbers can be implemented to run in time polynomial in the total number of bits in the input matrix [Edm67] and that each S_j is linearly independent, we could easily compute all S_j -canonical vectors (for all $j \in \{1, \dots, r\}$) in time $2^{O(k \log(ka) + k \log m)}$ by going through all candidate vectors $\mathbf{v} \in \{-ka, \dots, ka\}^k$ and checking whether there exist $0 \leq t_1, \dots, t_d < 1$ such that $\sum_{i=1}^d t_i \mathbf{u}_i = \mathbf{v}$. [Transform into row-reduced echelon form to compute the *unique* solution, if exists. Since $S_j \cup \{\mathbf{v}\} \subseteq \mathbb{Z}^k$, the coefficients t_1, \dots, t_d will be rational.]

For each fixed $j \in \{1, \dots, r\}$ and each fixed S_j -canonical vector \mathbf{v}_0 , we now show how to compute the set of all \preceq_j -minimal vectors \mathbf{v} such that $[\mathbf{v}] = \mathbf{v}_0$ by dynamic programming in time $2^{O(k \log m + k^2 \log(ka))}$. Observe that since there are at most $r(2ak + 1)^k = 2^{O(k \log(kam))}$ possible \mathbf{v}_0 , doing this for *all* canonical vectors would take time $2^{O(k \log m + k^2 \log(ka))}$, which is also the total complexity of the algorithm. To this end, we first fill out *in stages* a table T_1 which keeps track of all vectors $\mathbf{v} \in \{0, 1, \dots, N\}^k \cap \text{cone}_{\mathbb{N}}(V)$. At stage $h = 1, 2, \dots, m$, we collect all vectors \mathbf{v} that can be written as $\sum_{i=1}^h c_i \mathbf{v}_i$, where $0 \leq c_i \leq M$. Since the size of the table is at most $N^k(k \log N)$ — $k \log N$ bits are used to identify each element in the table with an associated k -tuple — this could be carried out in time $O(m(N^k(k \log N))^2) = 2^{O(k \log m + k^2 \log(ka))}$. We then fill out in stages another table T_2 , which keeps track of all vectors $\mathbf{v} \in \{0, 1, \dots, N\}^k \cap P(\mathbf{v}_0; S_j)$. This could be done in d stages, similar to the computation of T_1 , and could be implemented to run in time $O(k(N^k(k \log N))^2) = 2^{O(k \log m + k^2 \log(ka))}$. We then simply compute a new table $T_3 = T_1 \cap T_2$, from which we eliminate vectors that are not \preceq_j -minimal by using the characterization of \preceq_j -minimal vectors from Lemma 7.2.3. All in all, this could be implemented to run in time $2^{O(k \log m + k^2 \log(ka))}$.

7.3 Parikh images of regular languages

7.3.1 A normal form theorem

In this section, we shall apply Theorem 7.2.1 to obtain a normal form theorem for Parikh images of NWAs.

Theorem 7.3.1 *Let \mathcal{A} be an NWA with n states over an alphabet Σ of size k . Then, there exists a representation of the Parikh images $\mathcal{P}(\mathcal{L}(\mathcal{A}))$ of \mathcal{A} as a union of linear sets $P(\mathbf{v}_1; S_1), \dots, P(\mathbf{v}_m; S_m)$, where the maximum entry of each \mathbf{v}_i is $O(n^{3(k+1)}k^{4k+6})$, each S_i is a subset of $\{0, \dots, n\}^k$ with $|S_i| \leq k$, and $m = O(n^{k^2+3k+3}k^{4k+6})$. Furthermore, this is computable in time $2^{O(k^2 \log(kn))}$.*

Observe that this theorem causes an exponential blow-up only in the size of the alphabet. Efficiency could be improved by outputting numbers in binary.

We shall now prove this theorem. Let $\mathcal{A} = (\Sigma, Q, \delta, q_0, q_F)$ be a given NWA, where $|Q| = n$ and $\Sigma = \{a_1, \dots, a_k\}$. Throughout the proof, we shall use the notion of “cycle type”. A *cycle type* is a Parikh image $\mathbf{v} \in \mathbb{N}^k$ of any word $w \in \Sigma^{\leq n}$ such that there is a path π of \mathcal{A} on w from some (not necessarily initial) state p to itself. The cycle π is said to *witness* \mathbf{v} . Observe that the sum of the components of any cycle type cannot exceed n .

Characterization of $\mathcal{P}(\mathcal{L}(\mathcal{A}))$

We start with a characterization of the Parikh image of \mathcal{A} in terms of Parikh images of “short” paths together with some cycle types. Given a path $\pi = p_0 a_1 p_1 \dots a_r p_r$ of \mathcal{A} from the state p_0 to the state p_r , let $S_\pi \subseteq \{0, \dots, n\}^k$ be the set of all the cycle types that are witnessed by some cycles $C = p'_0 p'_1 \dots p'_t p'_0$ in \mathcal{A} such that $p'_i = p_j$ for some $i \in \{0, \dots, t\}$ and $j \in \{0, \dots, r\}$. That is, C and π *meet* at state $p'_i = p_j$; see Figure 7.1. Now define T_π to be the linear set $P(\mathcal{P}(\pi); S_\pi)$.

Lemma 7.3.2 *The following identity holds:*

$$\mathcal{P}(\mathcal{L}(\mathcal{A})) = \bigcup_{\pi} T_\pi,$$

where π is taken over all accepting runs of \mathcal{A} of length at most $(n-1)^2$.

To prove this lemma, we shall make use of the following simple fact, whose proof (in Appendix) is to a large extent similar to the well-known fact from graph theory that

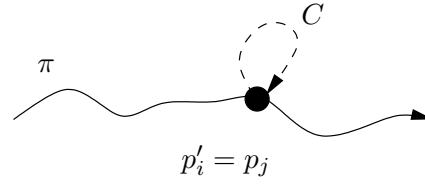


Figure 7.1: The path π (solid line) meets with the cycle C (broken line) at the state $p'_i = p_j$ (filled circle).

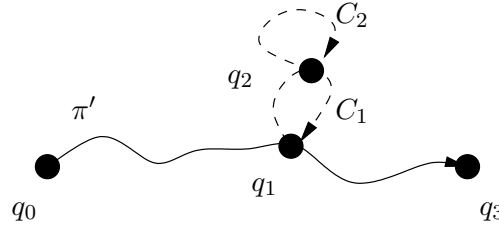


Figure 7.2: The path π is defined as follows: from q_0 it walks to q_1 , continues to q_2 (via half of cycle C_1), takes the cycle C_2 once, and then proceeds to q_1 (via the rest of cycle C_1) and straight to q_3 . The decomposition is π' (in solid line) and the cycles C_1 and C_2 (in broken lines).

the existence of a path between two given points implies the existence of a *simple* path between the same given points.

Fact 7.3.3 *Given an NWA \mathcal{A} with n states and a path π in \mathcal{A} from q to q' , there exist a simple path π' from q to q' and finitely many simple cycles C_1, \dots, C_h (possibly with duplicates) such that*

$$\mathcal{P}(\pi) = \mathcal{P}(\pi') + \sum_{i=1}^h \mathcal{P}(C_i).$$

An illustration of this simple fact is given in Figure 7.2.

Proof of Lemma 7.3.2. (\subseteq) Assume that $\mathbf{v} \in \mathcal{P}(\mathcal{L}(\mathcal{A}))$ and let $\sigma = p_0 a_1 p_1 \dots a_r p_r$ be an accepting run in \mathcal{A} such that $\mathcal{P}(\sigma) = \mathbf{v}$. We shall construct another accepting run σ' in \mathcal{A} of length at most $(n-1)^2$. For each state q occurring in σ , let $l(q)$ be the *last* (i.e. maximum) index $i \in \{0, \dots, r\}$ such that $p_i = q$. Let us write down all such $l(q)$ in an increasing order, e.g., $i_0 < i_1 < \dots < i_s = r$. Note that $s < n$. By Fact 7.3.3, each subpath $\sigma[i_j, i_{j+1}]$ of σ can be decomposed into a simple path π_j from p_{i_j} to $p_{i_{j+1}}$ of length at most $n-1$ and finitely many simple cycles (possibly with duplicates)

C_1, \dots, C_h each of length at most n such that $\mathcal{P}(\pi[i_j, i_{j+1}]) = \mathcal{P}(\pi_j) + \sum_{i=1}^h \mathcal{P}(C_i)$. Such a decomposition result, however, might allow some cycle C_i to *avoid* (i.e. not meet with) π_j . For example, in Figure 7.2 the cycle C_2 does not meet with the path π' . On the other hand, C_i *must* visit some states of p_{i_0}, \dots, p_{i_s} as this sequence contains all states in σ . Thus, we simply define σ' to be the accepting path $\pi_0 \odot \pi_1 \odot \dots \odot \pi_{s-1}$ of length at most $(n-1)^2$. It follows that $\mathbf{v} \in T_{\sigma'}$.

(\supseteq) Conversely, let $\mathbf{v} \in T_\pi$ for some accepting run π in \mathcal{A} of length at most $(n-1)^2$. Then, if $S_\pi = \{\mathbf{v}_1, \dots, \mathbf{v}_s\}$, then $\mathbf{v} = \mathcal{P}(\pi) + \sum_{i=1}^s t_i \mathbf{v}_i$ for some $t_1, \dots, t_s \in \mathbb{N}$. Let C_i be a cycle in \mathcal{A} that meets with π and satisfies $\mathcal{P}(C_i) = \mathbf{v}_i$. We can construct an accepting path σ in \mathcal{A} with $\mathcal{P}(\sigma) = \mathbf{v}$ as follows: start from π as the “base” path, and for each $i \in \{1, \dots, s\}$, attach t_i copies of C_i to one pre-selected common state of C_i and π . \square

As an immediate corollary of Lemma 7.3.2, we have:

Proposition 7.3.4 *Let \mathcal{A} be an NWA with n states over an alphabet Σ of size k . Then, $\mathcal{P}(\mathcal{L}(\mathcal{A}))$ can be represented as a union of linear sets $P(\mathbf{v}_1; S_1), \dots, P(\mathbf{v}_m; S_m)$, where $\mathbf{v}_i \in \{0, \dots, (n-1)^2\}^k$ and the components of each vector in S_i cannot exceed n .*

Remark: A slightly stronger version of this proposition was claimed in [SSM07], where the maximum component of each \mathbf{v}_i cannot exceed n . Their proof turns out to have a subtle error that also occurs in the proof of Chrobak-Martinez Theorem [Chr86, Mar02], which was recently fixed in [To09b]. In fact, we show in Proposition 7.3.10 below that our quadratic bound is essentially optimal, i.e., it cannot be lowered to $o(n^2)$. (End Remark)

Observe now that the proof of existence in Theorem 7.3.1 is essentially immediate from Proposition 7.3.4 and Theorem 7.2.1. We will next show that an algorithm for computing the desired semilinear basis can be obtained using a dynamic programming.

Dynamic programming algorithm

We first show how to compute all the cycle types of \mathcal{A} . More precisely, let $Q = \{q_0, \dots, q_{n-1}\}$, where $q_{n-1} := q_F$, and let $I = \{(t_1, \dots, t_k) \in \mathbb{N}^k : \sum_{i=1}^k t_i \leq n\}$. For each vector $\mathbf{v} \in \mathbb{N}^k$, we write $M_{\mathbf{v}} = [a_{i,j}]_{n \times n}$ for the n -by- n 0-1 matrix where $a_{i,j} = 1$ iff there exists a path π from q_i to q_j with $\mathcal{P}(\pi) = \mathbf{v}$. We are interested in computing all matrices $M_{\mathbf{v}}$ for each $\mathbf{v} \in I$. Observe that the naive algorithm, which runs through all paths of \mathcal{A} from q_i to q_j with $\mathcal{P}(\pi) = \mathbf{v}$, has time complexity that is exponential

in n . We will give an algorithm for computing these in time $2^{O(k \log n)}$ using dynamic programming. To this end, let us derive a recurrence relation for computing $M_{\mathbf{v}}$ based on $M_{\mathbf{v}'}$ with $\mathbf{v}' \preceq \mathbf{v}$. As a base case, we first observe that $M_{\mathbf{0}}$ is the n -by- n identity matrix. Furthermore, each matrix $M_{\mathbf{e}_i}$ could be constructed easily from the transition relation δ of \mathcal{A} . [Recall that $\{\mathbf{e}_i\}_{i=1}^k$ is the standard basis for \mathbb{R}^k .]

Lemma 7.3.5 *Let $\mathbf{v} = (r_1, r_2, \dots, r_{i-1}, r_i + 1, r_{i+1}, \dots, r_k)$ with each $r_i \in \mathbb{N}$. Then, the following identity holds:*

$$M_{\mathbf{v}} = \bigvee_{\mathbf{u}, \mathbf{w}} M_{\mathbf{u}} \bullet M_{\mathbf{e}_i} \bullet M_{\mathbf{w}}$$

where \mathbf{u} ranges over all vectors $\preceq \mathbf{v}$ whose i th entry is 0, and \mathbf{w} is the vector $\mathbf{v} - \mathbf{e}_i - \mathbf{u}$.

Intuitively, this recurrence relation can be derived by observing that a path π with $\mathcal{P}(\pi) = \mathbf{v}$ can be *uniquely* decomposed into three consecutive path segments π_1 , π_2 , and π_3 with $\mathcal{P}(\pi_1) = \mathbf{u}$, $\mathcal{P}(\pi_2) = \mathbf{e}_i$, and $\mathcal{P}(\pi_3) = \mathbf{w}$, for some \mathbf{u} and \mathbf{w} satisfying the prescribed condition. The path segment π_2 contains the *first* occurrence of the letter a_i in the path π . The proof of this lemma can be found in the appendix. The following lemma, whose proof is also in the appendix, is a simple application of Lemma 7.3.5 and dynamic programming.

Lemma 7.3.6 *We can compute $\{M_{\mathbf{v}}\}_{\mathbf{v} \in I}$ in time $2^{O(k \log n)}$.*

For each $i \in \{0, \dots, n-1\}$, let Γ_i be the set of all cycle types \mathbf{v} witnessed by some cycle $\pi = p_0 p_1 \dots p_0$ in \mathcal{A} with $p_j = q_i$. Since $\{M_{\mathbf{v}}\}_{\mathbf{v} \in I}$ have been computed, all sets Γ_i could be computed within $O(n^{k+1})$ extra time.

We now show how to compute $\mathcal{P}(\mathcal{L}(\mathcal{A}))$ in time $2^{O(k^2 \log(kn))}$. To this end, we shall use another application of dynamic programming based on Lemma 7.3.2, Theorem 7.2.1, and the sets $\{\Gamma_i\}_{i=0}^{n-1}$, which we already computed. For each $0 \leq i \leq (n-1)^2$ and each $0 \leq j < n$, let $T_{i,j} := \bigcup_{\pi} T_{\pi}$ where π is taken over all paths in \mathcal{A} of length i from q_0 to q_j . By Lemma 7.3.2, it is the case that $\mathcal{P}(\mathcal{L}(\mathcal{A})) = \bigcup_{i=0}^{(n-1)^2} T_{i,n-1}$; recall that $q_{n-1} = q_F$ by definition. We shall now derive a recurrence relation for $T_{i,j}$.

Lemma 7.3.7 *It is the case that $T_{0,0} = \{\mathbf{0}\}$ and $T_{0,j} = \emptyset$ for each $j \in \{1, \dots, n-1\}$. Whenever $i > 0$ and $j \in \{0, \dots, n-1\}$, we have*

$$T_{i,j} = \bigcup_{h=0}^{n-1} \left(T_{i-1,h} + \bigcup_{1 \leq l \leq k, (q_h, a_l, q_j) \in \delta} P(\mathbf{e}_l, \Gamma_j) \right).$$

This recurrence relation can be derived by observing that every path π of length i from q_0 to q_j can be decomposed into the path $\pi[0, i-1]$ ending at some state q_h and the path $\pi[i-1, i] = q_h a_l q_j$. The cycle types Γ_j can be “used” since q_j is visited. The proof is in the appendix.

To finish the proof of Theorem 7.3.1, it suffices to give an algorithm with running time $2^{O(k^2 \log(kn))}$ for computing a desired semilinear basis $P_{i,j}$ for each set $T_{i,j}$. The algorithm runs in $(n-1)^2 + 1$ stages, where at stage $i = 0, \dots, (n-1)^2$ the set $P_{i,j}$ is computed. Obviously, we first set $P_{0,0} = \{\mathbf{0}\}$ and $P_{0,j} = \emptyset$ for each $j \in \{1, \dots, n-1\}$. Inductively, suppose that $P_{i,h} = \{\langle \mathbf{v}_s^h; S_s^h \rangle\}_{s=1}^{m_h}$ has been computed for each $h \in \{0, \dots, n-1\}$. We will show how to compute $P_{i+1,j}$ for any given $j \in \{0, \dots, n-1\}$. For each $h \in \{0, \dots, n-1\}$, let J_h denote the set of numbers $l \in \{1, \dots, k\}$ such that $(q_h, a_l, q_j) \in \delta$. Therefore, we have $P_{i,h} + \bigcup_{l \in J_h} P(\mathbf{e}_l; \Gamma_j) = \bigcup_{s=1}^{m_h} \bigcup_{l \in J_h} P(\mathbf{v}_s^h + \mathbf{e}_l; S_s^h \cup \Gamma_j)$. We use the algorithm from Theorem 7.2.1 to compute another semilinear basis for each $P(\mathbf{v}_s^h + \mathbf{e}_l; S_s^h \cup \Gamma_j)$ and then compute unions in the obvious way to obtain $P_{i+1,j}$ (note: duplicates is removed). The output of this algorithm is $P = \bigcup_{i=1}^{(n-1)^2} P_{i,n-1}$. The correctness of the algorithm is immediate from Lemma 7.3.7.

We now analyze the time complexity of this algorithm. By induction, it is easy to see that at every stage of the algorithm $S_s^h \cup \Gamma_j \subseteq \{0, \dots, n\}^k$ holds for each $h \in \{0, \dots, n-1\}$ and $s \in \{1, \dots, m_h\}$. Therefore, the maximum component over all offsets in the semilinear basis $P_{i+1,j}$ is at most $a + O(n^{3(k+1)}k^{4k+6})$, where a is the maximum entry in each $\mathbf{v}_s^h + \mathbf{e}_l$ over all $h \in \{0, \dots, n-1\}$, $l \in J_h$, and $s \in \{1, \dots, m_h\}$. Note that the summand $O(n^{3(k+1)}k^{4k+6})$ is due to an application of Theorem 7.2.1. By induction, at stage i the maximum component over all offsets in $\{P_{i,j}\}_{j=0}^{n-1}$ is $i \times O(n^{3(k+1)}k^{4k+6})$. This means that the maximum entry of each offset in P is $O(n^{3k+5}k^{4k+6})$, and the number of linear bases in P is $O(n^{k^2+3k+5}k^{4k+6})$ (since duplicates are always removed). It is also easy to see that at each stage i , the algorithm runs in time $2^{O(k^2 \log(nk))}$, primarily spent in the algorithm from Theorem 7.2.1. All in all, our algorithm runs in time $2^{O(k^2 \log(nk))}$, which is also the complexity of the entire procedure.

7.3.2 Complementary lower bounds

We shall now prove three lower bounds to complement earlier results in this section. We start by proving that *every* semilinear basis for the Parikh image of a DWA can be large in the size of the alphabet.

Proposition 7.3.8 *For each $k \in \mathbb{Z}_{>0}$ and each integer $n > 1$, there exists a DWA $\mathcal{A}_{n,k}$*

over the alphabet $\Sigma_k := \{a_1, \dots, a_k\}$ with $n + 1$ states whose Parikh image contains at least $n^{k-1}/(k-1)!$ linear sets.

Proof. Let $\mathcal{A}_{n,k} = (Q = \{q_0, \dots, q_n\}, \delta, q_0, q_n)$ with $\delta(q_i, a) = q_{i+1}$ for each $a \in \Sigma_k$ and $0 \leq i < n$. This automaton has a finite language $\mathcal{L}(\mathcal{A}_{n,k})$ with Parikh image $\mathcal{P}(\mathcal{L}(\mathcal{A}_{n,k}))$ containing precisely all *ordered integer partitions* of n into k parts, i.e., all tuples (n_1, \dots, n_k) with $\sum_{i=1}^k n_i = n$. Since the set $\mathcal{P}(\mathcal{L}(\mathcal{A}_{n,k}))$ is finite, each ordered integer partition (n_1, \dots, n_k) of n must appear in precisely one linear set. Finally, it is easy to check (e.g. see [vLW01, Chapter 13]) that the number of ordered partitions of n into k parts equals $\binom{n+k-1}{k-1} \geq n^{k-1}/(k-1)!$. \square

This proposition implies that, for every fixed $k \geq 1$, there exists infinitely many DWAs $\{\mathcal{A}_n\}$ over an alphabet of size k where \mathcal{A}_n has size $O(n)$ but $\mathcal{P}(\mathcal{L}(\mathcal{A}_n))$ must contain $\Omega(n^{k-1})$ linear bases. Therefore, this shows that k *cannot* be removed from the exponent in Theorem 7.3.1. In addition, observing that the DWAs that we constructed have equivalent regular expressions of size $O(n)$, Proposition 7.3.8 also gives lower bounds for Parikh images of regular expressions.

Next, we show that Theorem 7.3.1 *cannot* be extended to languages of CFGs (equivalently, PDAs). More precisely, we show that the number of linear sets for Parikh images of CFGs could be exponential in the size of the CFGs.

Proposition 7.3.9 *There exists a small constant $c \in \mathbb{Z}_{>0}$ such that, for each integer $n > 1$, there exists a CFG \mathcal{G}_n of size at most cn over the alphabet $\Sigma := \{a\}$ whose Parikh image contains precisely 2^n linear sets.*

Proof. We will construct a CFG Σ_n such that $\mathcal{P}(\mathcal{L}(\mathcal{G}_n)) = \{0, 1, \dots, 2^n - 1\}$, each of whose elements will appear in precisely one linear set. Our construction uses the lower bound technique in [PSW02].

Our CFG \mathcal{G}_n contains nonterminals $S, \{A_i\}_{i=0}^{n-1}$, and $\{B_i\}_{i=0}^{n-1}$, and consists precisely of the following rules:

$$\begin{aligned} S &\rightarrow A_0 \dots A_{n-1} \\ A_i &\rightarrow \varepsilon \quad \text{for each } 0 \leq i < n \\ A_i &\rightarrow B_i \quad \text{for each } 0 \leq i < n \\ B_i &\rightarrow B_{i-1} B_{i-1} \quad \text{for each } 0 < i < n \\ B_0 &\rightarrow a \end{aligned}$$

The initial nonterminal is declared to be S . It is easy to prove by induction that, for each word $w \in \Sigma^*$, $B_i \Rightarrow^* w$ iff $w = a^{2^i}$. This implies that A_i generates either ε or a^{2^i} . Thus, we see that $\mathcal{L}(\mathcal{G}_n) = \{a^i : 0 \leq i < 2^n\}$, which easily yields the desired result. \square

Finally, we give a lower bound proving the tightness of quadratic upper bound in Proposition 7.3.4, even when restricted to DWAs.

Proposition 7.3.10 *For each positive integer $n > 2$, there exists a DWA \mathcal{A}_n with $2n + 3$ states such that if $\mathcal{P}(\mathcal{L}(\mathcal{A}_n)) = \bigcup_{i=1}^r P(\mathbf{v}_i; S_i)$, then one entry in some \mathbf{v}_i is at least $n(n + 1)/2$.*

This proof is given in the appendix. In fact, the constructed DWA \mathcal{A}_n has an equivalent regular expression of size $O(n)$ as well, therefore yielding the same quadratic lower bound for regular expressions.

7.4 Three simple applications

In this section, we shall give three simple applications of our main results in previous sections not all of which are related to model checking: (1) polynomial-time fragments of integer linear programming, (2) decision problems for Parikh images of NWAs, and (3) Presburger-constrained graph reachability. As we shall see in the next section, some of these results will be used to obtain better complexity upper bounds for model checking over reversal-bounded counter systems and their extensions with discrete clocks. Other applications of the main results in the previous sections including polynomial PAC-learnability of semilinear sets (with unary representation of numbers) can be found in [To10].

7.4.1 Integer programming

Integer programming (IP) is the problem of checking whether a given integer program $A\mathbf{x} = \mathbf{b}$ ($\mathbf{x} \succeq \mathbf{0}$), where A is a k -by- m integer matrix and $\mathbf{b} \in \mathbb{Z}^k$, has an integral solution. This problem is a standard NP-complete problem in computational complexity (cf. [PS98]). We shall mention two well-known polynomial-time fragments of IP and then give a generalization that subsumes both.

The first polynomial-time fragment of IP, due to Lenstra [Len83], is obtained by fixing the number m of variables in the integer programs. More precisely, Lenstra showed that IP is solvable in time polynomial in the size of the input and exponential

in m . The complexity of Lenstra's algorithm has been improved by Kannan [Kan83] to $m^{\log m} L \log L$, where L is the input length. Let us now mention the second polynomial-time fragment, due to Papadimitriou [Pap81]. Firstly, observe that when $k = 1$ is fixed, IP reduces to the well-known NP-complete knapsack problem (cf. [PS98]). On the other hand, the knapsack problem is easily seen to be pseudopolynomial-time solvable (cf. [PS98]), i.e., solvable in polynomial-time when numbers in the input are represented in unary. The second polynomial-time fragment of IP is obtained by restricting the number k of equations in the integer programs and enforcing the numbers in the input to be represented in unary. Therefore, it is a generalization of the knapsack problem when numbers in the input are represented in unary. Papadimitriou [Pap81] gave an algorithm for solving IP that runs in time $2^{O(k \log m + k^2 \log(ka))}$, where a is the maximum absolute value of numbers appearing in the input. This immediately yields a polynomial-time algorithm for the second fragment of IP. Notice that the complexity of the algorithm is exponential unless a is represented in unary and k is fixed.

We now present a generalization of the two aforementioned polynomial-time fragments of IP. Let $\text{IP}_{k,m}$ be the problem of deciding whether an integer program $\mathbf{Ax} = \mathbf{b}$ has a non-negative integral solution, where $\mathbf{b} \in \mathbb{Z}^{k'}$ is represented in unary and A is a k' -by- m' integer matrix of the form

$$A = \left[\begin{array}{c|c} A_1 & A_2 \\ \hline A_3 & 0 \end{array} \right] \quad (7.1)$$

for a k -by- m matrix A_1 and a $(k' - k)$ -by- m matrix A_3 where numbers are represented in binary, and a k -by- $(m' - m)$ matrix A_2 where numbers are represented in unary. The bottom right block of A is simply a $(k' - k)$ -by- $(m' - m)$ matrix full of zeros.

Proposition 7.4.1 *For fixed integers $k > 0$ and $m > 0$, the problem $\text{IP}_{k,m}$ is solvable in polynomial-time.*

Proof. Let $\mathbf{Ax} = \mathbf{b}$ be the given integer program, where A is of the form given in Equation 7.1 above. Let $m_1 = m' - m$. Write the given integer program $\mathbf{Ax} = \mathbf{b}$ in an equational form:

$$\begin{array}{ccccccccccc} a_{1,1}x_1 & + & \dots & + & a_{1,m}x_m & + & c_{1,1}y_1 & + & \dots & + & c_{1,m_1}y_{m_1} & = & b_1 \\ \vdots & & \ddots & & \vdots & & \vdots & & \ddots & & \vdots & & \vdots \\ a_{k,1}x_1 & + & \dots & + & a_{k,m}x_m & + & c_{k,1}y_1 & + & \dots & + & c_{k,m_1}y_{m_1} & = & b_k \\ a_{k+1,1}x_1 & + & \dots & + & a_{k+1,m}x_m & & & & & & & = & b_{k+1} \\ \vdots & & \ddots & & \vdots & & & & & & & & \vdots \\ a_{k',1}x_1 & + & \dots & + & a_{k',m}x_m & & & & & & & = & b_{k'}. \end{array}$$

For each $i = 1, \dots, m_1$, let us write \mathbf{c}_i for the i th column vector of A_2 , i.e., the transpose of the row vector $(c_{1,i}, \dots, c_{k,i})$. Let $V = \{\mathbf{c}_1, \dots, \mathbf{c}_{m_1}\}$ and denote by a be the maximum absolute value of numbers in V . By Theorem 7.2.1, we may compute in time $2^{O(k \log m_1 + k^2 \log(ka))}$ a sequence $P(\mathbf{w}_1; S_1), \dots, P(\mathbf{w}_r; S_r)$ of linear sets with

$$\mathbf{cone}_{\mathbb{N}}(V) = \bigcup_{i=1}^r P(\mathbf{w}_i; S_i),$$

where the maximum absolute value of entries of each \mathbf{w}_i is $O(m_1(k^2a)^{2k+3})$, each S_i is a subset of V with $|S_i| = \mathbf{rank}(V) \leq k$, and $r = O(m_1^{2k}(k^2a)^{2k^2+3k})$. Let $d = \mathbf{rank}(V)$, which we can compute in polynomial time using Gaussian elimination. Therefore, $A\mathbf{x} = \mathbf{b}$ has a non-negative integral solution iff for some $P(\mathbf{w}_j; S_j)$, say with $\mathbf{w}_j = (s_1, \dots, s_k)$ and $S_j = \{\mathbf{c}_{i_1}, \dots, \mathbf{c}_{i_d}\}$, the following integer program P_j in equational form has a non-negative integral solution:

$$\begin{array}{ccccccccccc} a_{1,1}x_1 & + & \dots & + & a_{1,m}x_m & + & c_{1,i_1}y_1 & + & \dots & + & c_{1,i_d}y_d & = & b_1 - s_1 \\ \vdots & & \ddots & & \vdots & & \vdots & & \ddots & & \vdots & & \vdots \\ a_{k,1}x_1 & + & \dots & + & a_{k,m}x_m & + & c_{k,i_1}y_1 & + & \dots & + & c_{k,i_d}y_d & = & b_k - s_k \\ a_{k+1,1}x_1 & + & \dots & + & a_{k+1,m}x_m & & & & & & & = & b_{k+1} \\ \vdots & & \ddots & & \vdots & & & & & & & & \vdots \\ a_{k',1}x_1 & + & \dots & + & a_{k',m}x_m & & & & & & & = & b_{k'}. \end{array}$$

Note that the size of each \mathbf{w}_j when the numbers are given in binary representation is at most kL , where L is the size of the original integer program. Hence, the size of the integer program P_j is at most kL . Therefore, Kannan's algorithm [Kan83] can solve each integer program P_j in time $O((m+k)^{9(m+k)}kL \log(kL))$, where L is the size of the input. In the worst case, we will have to run Kannan's algorithm on each P_j . All in all, the total running time is

$$2^{O(k \log m_1 + k^2 \log(ka))} + O(m_1^{2k}(k^2a)^{2k^2+3k} \times (m+k)^{9(m+k)}kL \log(kL)),$$

which is polynomial when k and m are fixed constants, and a is represented in unary.

□

7.4.2 Decision problems for Parikh images of NWAs

We now give another application of the main results from the previous sections to the following decision problems for Parikh images of NWAs: membership, disjointness, universality, and equivalence. The result for membership was independently proven by

Kopczynski [Kop10] and the author [To10]. The results for disjointness, universality, and equivalence were initially shown by Kopczynski [Kop10], different proofs are given below.

Membership

The *membership problem for Parikh images of NWAs* is defined as follows: given an NWA \mathcal{A} over $\Sigma = \{a_1, \dots, a_k\}$ and a tuple $\mathbf{b} \in \mathbb{N}^k$ given in binary, decide whether $\mathbf{b} \in \mathcal{P}(\mathcal{L}(\mathcal{A}))$. Similar problems can be easily defined for DWAs, regular expressions, CFGs, and PDAs. It is known that the membership problem for Parikh images of NWAs is solvable in NP (e.g. see [Esp97b, Huy83, VSS05]). It turns out that this upper bound is tight.

Proposition 7.4.2 *The membership problems for Parikh images of DWAs and regular expressions are NP-hard, even when numbers in the input are given in unary.*

The proof is given in the appendix. The lower bound for DWAs is obtained by a reduction from the well-known NP-complete hamiltonian path problem. On the other hand, NP-hardness for regular expressions is obtained by a reduction from a variant of 3SAT. The lower bound for DWAs was also independently proven by Kopczynski [Kop10].

In contrast to Proposition 7.4.2, the membership problem for Parikh images of NWAs becomes solvable in polynomial time when the size k of the alphabet is fixed.

Proposition 7.4.3 ([Kop10, KT10, To10]) *Given an NWA \mathcal{A} with n states over the alphabet $\Sigma = \{a_1, \dots, a_k\}$ and a tuple $\mathbf{b} = (b_1, \dots, b_k) \in \mathbb{N}^k$ written in binary with $b := \max_{1 \leq i \leq k} \{b_i\}$, checking whether $\mathbf{b} \in \mathcal{P}(\mathcal{L}(\mathcal{A}))$ can be done in time $2^{O(k^2 \log(kn) + \log \log b)}$.*

This proposition can be obtained almost in the same way as in Proposition 7.4.1. That is, we first use Theorem 7.3.1 to compute a union of linear sets $P(\mathbf{w}_1; S_1), \dots, P(\mathbf{w}_r; S_r)$ with at most k periods that such that $\mathcal{P}(\mathcal{L}(\mathcal{A})) = \bigcup_{i=1}^r P(\mathbf{w}_i; S_i)$. Then, we simply need to test whether there exists $i = 1, \dots, r$ such that $\mathbf{b} \in P(\mathbf{w}_i; S_i)$, which can be done by Kannan's polynomial-time algorithm (as in Proposition 7.4.1).

Let us finally remark that Proposition 7.4.3 cannot be extended to CFGs (or equivalently PDAs).

Proposition 7.4.4 ([Kop10, KT10, To10]) *The membership problem of CFGs over the alphabet $\Sigma = \{a\}$ is NP-hard.*

This proposition improves the known NP-hardness lower bound for the problem over a non-fixed alphabet (e.g. see [Esp97b, Huy83]). The proof of this proposition is by a simple reduction from the knapsack problem using the succinct encoding of numbers given in the proof of Proposition 7.3.9. In fact, the lower bound is optimal since the membership problem of CFGs is solvable in NP [Esp97b, Huy83].

Disjointness

The *disjointness problem for Parikh images of NWAs* is defined as follows: given two NWAs \mathcal{A} and \mathcal{B} over $\Sigma = \{a_1, \dots, a_k\}$, decide whether $\mathcal{P}(\mathcal{L}(\mathcal{A})) \cap \mathcal{P}(\mathcal{L}(\mathcal{B})) = \emptyset$. Similar problems can be easily defined for CFGs. First of all, it is a simple corollary of the result of [VSS05] that the disjointness problem for Parikh images of CFGs is in coNP. This is because there exists a polynomial time algorithm which, given two CFGs G_1 and G_2 over the alphabet $\Sigma = \{a_1, \dots, a_k\}$, computes an existential Presburger formula $\varphi_1(x_1, \dots, x_k)$ and $\varphi_2(x_1, \dots, x_k)$ such that, for each $i = 1, 2$ and numbers $m_1, \dots, m_k \in \mathbb{N}$,

$$\langle \mathbb{N}, + \rangle \models \varphi_i(m_1, \dots, m_k) \Leftrightarrow (m_1, \dots, m_k) \in \mathcal{P}(\mathcal{L}(G_i)).$$

Testing whether $\mathcal{P}(\mathcal{L}(G_1)) \cap \mathcal{P}(\mathcal{L}(G_2)) \neq \emptyset$ then corresponds to checking whether

$$\langle \mathbb{N}, + \rangle \models \exists x_1, \dots, x_k (\varphi_1(x_1, \dots, x_k) \wedge \varphi_2(x_1, \dots, x_k)),$$

which can be done in NP since checking existential Presburger formulas is in NP [GS78]. It follows that checking whether $\mathcal{P}(\mathcal{L}(G_1)) \cap \mathcal{P}(\mathcal{L}(G_2)) = \emptyset$ is in coNP. In fact, a matching coNP lower bound has been shown by Kopczynski [Kop10] even for CFGs over the fixed alphabet $\{a\}$. This simple proof technique also gives an easy polynomial-time upper bound for disjointness problem for Parikh images of NWAs for a fixed alphabet size, which was first shown by Kopczynski [Kop10] using a different technique, i.e., by observing that Theorem 7.3.1 holds for NWAs with negative inputs.

Proposition 7.4.5 *The disjointness problem for Parikh images of NWAs over a fixed alphabet $\Sigma = \{a_1, \dots, a_k\}$ can be decided in polynomial-time.*

We now sketch a proof of this proposition. Given two NWAs $\mathcal{A}_1, \mathcal{A}_2$ over Σ with (respectively) n_1 and n_2 states, we may apply Theorem 7.3.1 on \mathcal{A}_1 and \mathcal{A}_2 to obtain

in polynomial time two semilinear bases $\mathcal{B}_1 := \{\langle \mathbf{v}_1; S_1 \rangle, \dots, \langle \mathbf{v}_{m_1}; S_{m_1} \rangle\}$ and $\mathcal{B}_2 := \{\langle \mathbf{w}_1; S'_1 \rangle, \dots, \langle \mathbf{w}_{m_2}; S'_{m_2} \rangle\}$ for, respectively, the Parikh images of $\mathcal{L}(\mathcal{A}_1)$ and $\mathcal{L}(\mathcal{A}_2)$ such that $|S_i| \leq k$ and $|S'_j| \leq k$. We shall now state an easy fact relating semilinear bases and existential Presburger formulas.

Fact 7.4.6 *Given a semilinear basis $\mathcal{B} = \{\langle \mathbf{v}_1; S_1 \rangle, \dots, \langle \mathbf{v}_r; S_r \rangle\}$ over \mathbb{N}^k with $|S_i| = m$ for each $1 \leq i \leq r$, there exists an existential Presburger formula $\varphi(x_1, \dots, x_k)$ of the form*

$$\varphi(x_1, \dots, x_k) = \exists y_1, \dots, y_m \psi(\bar{x}, \bar{y})$$

such that ψ is quantifier-free and, for each sequence i_1, \dots, i_k of nonnegative integers, it is the case that

$$\langle \mathbb{N}, + \rangle \models \varphi(i_1, \dots, i_k) \Leftrightarrow (i_1, \dots, i_k) \in P(\mathcal{B}).$$

Furthermore, $\|\varphi\|$ is linear in $\|\mathcal{B}\|$ (even with binary representation of numbers) and that φ can be computed in linear time.

Example 7.4.1 We shall give an example of how the translation from Fact 7.4.6 is performed. Suppose we have the semilinear basis

$$\mathcal{B} = \{\langle (10, 3); \{(1, 2), (8, 7)\} \rangle, \langle (5, 5); \{(7, 1), (25, 13)\} \rangle\}.$$

The existential Presburger formula that represents $P(\mathcal{B})$ is simply

$$\varphi(x_1, x_2) := \exists y_1, y_2 (\psi_1(\bar{x}, \bar{y}) \vee \psi_2(\bar{x}, \bar{y})),$$

where

$$\psi_1(\bar{x}, \bar{y}) := (x_1 = 10 + y_1 + 8y_2) \wedge (x_2 = 3 + 2y_1 + 7y_2)$$

and

$$\psi_2(\bar{x}, \bar{y}) := (x_1 = 5 + 7y_1 + 25y_2) \wedge (x_2 = 5 + y_1 + 13y_2).$$



Therefore, we compute existential Presburger formulas $\varphi_1(x_1, \dots, x_k)$ and $\varphi_2(x_1, \dots, x_k)$ each with at most k quantifiers, which represent \mathcal{B}_1 and \mathcal{B}_2 , respectively. Hence, we have

$$P(\mathcal{B}_1) \cap P(\mathcal{B}_2) \neq \emptyset \Leftrightarrow \langle \mathbb{N}, + \rangle \models \exists x_1, \dots, x_k (\varphi_1(\bar{x}) \wedge \varphi_2(\bar{x})).$$

Clearly, we can move the existential quantifiers in φ_1 and φ_2 to the front of the formula (after introducing new variable names) yielding an existential formula θ with $3k$ quantifiers. That is, the formula θ has a fixed number of quantifiers regardless of the input

automata \mathcal{A}_1 and \mathcal{A}_2 . Since checking existential Presburger formulas with a fixed number of quantifiers can be done in polynomial time [Len83, Sca84] (also see [Grä88]), it follows that checking $P(\mathcal{B}_1) \cap P(\mathcal{B}_2) \neq \emptyset$ can be done in polynomial time. Proposition 7.4.5 immediately follows.

As a final remark, Proposition 7.4.5 is tight in the sense that allowing unbounded alphabet size makes the problem coNP-complete [Kop10].

Universality and equivalence

The *universality problem for Parikh images of NWAs* is defined as follows: given an NWA \mathcal{A} over $\Sigma = \{a_1, \dots, a_k\}$, decide whether $\mathcal{P}(\mathcal{L}(\mathcal{A})) = \mathbb{N}^k$. The *equivalence problem for Parikh images of NWAs* is defined as follows: given two NWAs \mathcal{A}_1 and \mathcal{A}_2 over $\Sigma = \{a_1, \dots, a_k\}$, decide whether $\mathcal{P}(\mathcal{L}(\mathcal{A}_1)) = \mathcal{P}(\mathcal{L}(\mathcal{A}_2))$. Kopczynski [Kop10] was the first to observe that Theorem 7.3.1 yields coNP upper bounds for these two problems in the case of a fixed alphabet size. In fact, there is a matching coNP lower bound for these two problems even in the case of alphabet of size 1, which was first shown by Stockmeyer and Meyer [SM73]. The precise complexity in the case of unbounded alphabet size is only known to be in between coNEXP and coNP [Kop10, KT10]. In the following, we shall employ the same technique that we use for the disjointness problem above to rederive Kopczynski's coNP upper bound for universality and equivalence.

Proposition 7.4.7 *Universality and equivalence problems for Parikh images of NWAs over a fixed alphabet $\{a_1, \dots, a_k\}$ are in coNP.*

Our proof is similar to the proof of Proposition 7.4.5, but instead uses Grädel's result [Grä88] that evaluating Presburger formulas of the form $\forall \bar{x} \exists \bar{y} \psi(\bar{x}, \bar{y})$ is coNP-complete provided that the number of variables in \bar{x} and \bar{y} is fixed. For example, for the universality problem, we invoke the algorithm from Theorem 7.3.1 to compute in polynomial time a semilinear basis (each of whose linear bases has at most k periods) that represents the Parikh image $\mathcal{P}(\mathcal{L}(\mathcal{A}))$ of the given automaton \mathcal{A} . Using Fact 7.4.6, we may then compute an existential Presburger formula $\phi(x_1, \dots, x_k)$ with k quantifiers that represents $\mathcal{P}(\mathcal{L}(\mathcal{A}))$. Checking universality of $\mathcal{P}(\mathcal{L}(\mathcal{A}))$ then amounts to checking whether $\langle \mathbb{N}, + \rangle \models \forall \bar{x} \phi(x_1, \dots, x_k)$, which is in coNP by [Grä88] since the number of quantifiers in this formula is at most $2k$. The coNP upper bound for the equivalence problem can be derived in the same manner.

7.4.3 Presburger-constrained graph reachability

Presburger-constrained graph reachability is a simple extension of the standard graph reachability problem: given a transition system $\mathfrak{S} = \langle S, \{\rightarrow_a\}_{a \in \text{ACT}} \rangle$ over the action alphabet $\text{ACT} = \{a_1, \dots, a_k\}$, two configurations $s, t \in S$, and an arbitrary Presburger formula (a.k.a. *constraint*) $\phi(x_{i_1}, \dots, x_{i_r})$ (with $1 \leq i_1 < \dots < i_r \leq k$), decide whether there exists a path

$$s_0 \rightarrow_{a_{i_1}} \dots \rightarrow_{a_{i_m}} s_m$$

such that, if $(i_1, \dots, i_k) = \mathcal{P}(a_{i_1} \dots a_{i_m})$, then $\langle \mathbb{N}, + \rangle \models \phi(i_1, \dots, i_k)$. Problems related to Presburger-constrained graph reachability have been studied in the context of path-queries over graph-structured databases (cf. [BHLW10, HPW09b, HPW09a, MW95]). Presburger constraints are natural since in many cases we do not care about the order of actions in the path we are interested in. For example, consider a graph which models a transportation network, where the vertices are locations in the network (e.g. city attractions, bus stops, and so forth) and the edges are labeled by *means of transport* (e.g. bus, walk, airplane, subway). The reader is referred to [HPW09b, Figure 1] for a specific example. One could, for example, be interested in reaching a point t from a point s in such a graph by taking at most one subway and avoiding buses altogether. Observe that the order in which the actions take place is not important for such a query, which can therefore be expressed as a Presburger formula.

In general, Presburger-constrained graph reachability has the same complexity as evaluating Presburger formulas.

Proposition 7.4.8 *Presburger-constrained graph reachability is complete for the class $\text{STA}(*, 2^{2^{n^{O(1)}}}, n)$.*

To show this proposition, recall that the precise complexity of evaluating Presburger formulas is precisely $\text{STA}(*, 2^{2^{n^{O(1)}}}, n)$, due to Berman [Ber80] (also see [Koz06]). To derive the upper bound in Proposition 7.4.8, suppose that the input transition system is \mathfrak{S} over the action alphabet $\text{ACT} = \{a_1, \dots, a_k\}$ with initial configuration $s \in S$ and final configuration $t \in S$. We treat \mathfrak{S} as an NWA \mathcal{A} with initial state s and final state t , for which we can compute an existential Presburger formula $\psi(x_1, \dots, x_k)$ for $\mathcal{P}(\mathcal{L}(\mathcal{A}))$ in polynomial time using the result of [VSS05]. If the input Presburger formula is $\phi(x_{i_1}, \dots, x_{i_r})$, then the problem reduces to checking whether

$$\langle \mathbb{N}, + \rangle \models \exists x_1, \dots, x_k (\psi(x_1, \dots, x_k) \wedge \phi(x_{i_1}, \dots, x_{i_r})),$$

which can be solved by the procedure for evaluating Presburger formulas. This immediately yields the desired upper bound. Hardness for $STA(*, 2^{2^{O(1)}}, n)$ can be easily obtained by a polynomial reduction from the evaluation of Presburger formulas. In fact, hardness easily holds even for a fixed transition system over the action alphabet $\Sigma = \{a\}$ with one configuration (both of which are initial and final).

The high complexity in Proposition 7.4.8 can be lowered by, for example, considering only existential Presburger formulas, in which case the complexity becomes NP-complete (cf. [BHLW10]). Another way of lowering the complexity in Proposition 7.4.8 is by observing that the Presburger constraint is usually small in real life, i.e., practically fixed. In this case, it turns out that the complexity of Presburger-constrained graph reachability becomes polynomial-time solvable.

Proposition 7.4.9 *Presburger-constrained graph reachability is polynomial-time solvable for any fixed Presburger constraint.*

Proof. Suppose that the fixed Presburger constraint is $\varphi(x_1, \dots, x_r)$. Then, the classical result of Ginsburg and Spanier [GS66] says that there exists a semilinear basis \mathcal{B} such that, for all $(i_1, \dots, i_r) \in \mathbb{N}^k$, it is the case that

$$\langle \mathbb{N}, + \rangle \models \varphi(i_1, \dots, i_r) \iff (i_1, \dots, i_r) \in P(\mathcal{B}).$$

Therefore, using Fact 7.4.6, we may assume an existential Presburger formula $\varphi'(\bar{x})$ that is equivalent with $\varphi(\bar{x})$. Let h be the (fixed) number of quantifiers of φ' .

Given a transition system $\mathfrak{S} = \langle S, \{\rightarrow_a\}_{a \in \text{ACT}} \rangle$ over $\text{ACT} = \{a_1, \dots, a_k\}$ (where $r < k$), an initial configuration $s \in S$, and a final configuration $t \in S$, let $\text{ACT}' = \{a_1, \dots, a_r, ?\}$ and define the automaton $\mathcal{A} = (\text{ACT}', Q, \delta, q_0, q_F)$ as follows:

- $Q = S$,
- $\delta = (\bigcup_{i=1}^r \rightarrow_{a_i}) \cup \{(q, ?, q') : \exists j \in (r, k] (q \rightarrow_{a_j} q')\}$,
- $q_0 = s$, and
- $q_F = t$.

In other words, we relabel the action a_j ($j = r + 1, \dots, k$) with the new action symbol '?'. Since r is fixed, the NWA \mathcal{A} has a fixed alphabet size. Therefore, using Theorem 7.3.1 and Fact 7.4.6, we obtain in polynomial time an existential Presburger formula

$\Psi(x_1, \dots, x_{r+1})$ with a fixed number $r + 1$ of quantifiers for the Parikh image $\mathcal{P}(\mathcal{L}(\mathcal{A}))$. Hence, the input $\langle \mathcal{G}, s, t \rangle$ is a positive instance of the problem iff

$$\langle \mathbb{N}, + \rangle \models \exists x_1, \dots, x_{r+1} (\Psi(x_1, \dots, x_{r+1}) \wedge \Phi'(x_1, \dots, x_r)).$$

By moving the existential quantifiers to the front of the formula (after variable renaming), we obtain an existential formula with a fixed number $2r + 2 + h$ of quantifiers and of size polynomial in the size of the input (recall that Φ' has a fixed size), which can be evaluated in polynomial time by Lenstra-Scarpellini's algorithm [Len83, Sca84] (also see [Grä88]). \square

7.5 Applications to model checking

In this section, we shall show how the earlier results in this chapter can be used to obtain better model checking complexities for reversal-bounded counter systems and their extensions with discrete clocks.

7.5.1 LTL with complex fairness

We shall first state our main result of LTL model checking with multi-regular fairness constraints over reversal-bounded counter systems with discrete clocks.

Theorem 7.5.1 *Model checking an LTL formula Φ with multi-regular fairness constraints over r -reversal bounded k -counter system \mathcal{M} with n states and t discrete clocks can be done in time polynomial in the size of each fairness constraint and in n , but exponential in the following parameters: the number of fairness constraints, r , k , t , $\|\Phi\|$, and the size (in binary) of the maximum absolute value of clock comparison constant l in \mathcal{M} .*

Observe that this theorem improves Theorem 6.2.10 by a single exponential for almost all parameters at the expense of not allowing a single free counter. In fact, the time complexity of this algorithm cannot be improved for the following reasons. If we fix all the parameters but Φ and n , we obtain LTL model checking over finite systems, which is PSPACE-complete [SC85, VW86a]. If we fix all the parameters but t and n , we may still reduce in polynomial time the emptiness of discrete-timed automata, which are PSPACE-complete [CY92]. The same goes if we fix all parameters but l and t , since emptiness of timed automata is PSPACE-complete already for three clocks

[CY92]. What if we fix all parameters except for only n and r or only n and k ? It turns out that this problem is still NP-hard, as can be shown by a polynomial reduction from the NP-complete emptiness problem of reversal-bounded counter automata with a fixed number of reversals or with a fixed number of counters [HR87].

Let us now proceed to the proof of Theorem 7.5.1. This theorem can be proved by simply observing that Theorem 7.3.1 can be used to refine Proposition 6.2.6 when there is no free counter:

Proposition 7.5.2 *We can compute a representation of the Parikh image of the language of a given r -reversal bounded k -counter automaton \mathcal{M} with n states as a union of conjunctive queries over $\langle \mathbb{N}, + \rangle$ with linear equations consisting of at most $O(rk)$ summands (numbers are given in unary) such that each conjunctive query has at most $O(rk)$ conjuncts and $O(rk)$ variables. Furthermore, this can be done in time polynomial in n , and exponential in k and r .*

Notice that the complexity in this proposition is the same as the complexity in Proposition 6.2.6, but the output existential Presburger formulas have some further structures, which we have already seen in Fact 7.4.6. The crucial component of Proposition 7.5.2 is that the number of conjuncts and the number of existential quantifiers in every conjunctive query is at most $O(rk)$ and does not depend on the number n of states of \mathcal{M} . To prove this theorem, one simply replaces the use of the polynomial-time algorithm from [VSS05] in the modified Ibarra's algorithm by Theorem 7.3.1 following a slightly more precise analysis of the structure of existential Presburger formulas obtained in Fact 7.4.6 (i.e. by using the fact that existential quantifiers distribute across disjunctions). The following proposition is now a direct corollary of Proposition 7.5.2, Proposition 3.1.2, Proposition 6.2.7, and Proposition 6.2.9.

Proposition 7.5.3 *The reachability relation of a given n -state r -reversal bounded k -counter systems \mathcal{M} with t discrete clocks is regular, for which an NWA can be computed in time polynomial in n , but exponential in r , k , t , and the size (in binary) of the maximum absolute value of the clock comparison constants in \mathcal{M} .*

Combining this proposition with our algorithmic metatheorem for decidable LTL model checking over word-automatic systems, Theorem 7.5.1 is now immediate.

7.5.2 Branching-time logics

We now apply Theorem 7.3.1 to EF-model checking over reversal-bounded counter systems. Our main theorem is a kind of fixed-parameter tractability result:

Theorem 7.5.4 *Fix an EF-logic formula ϕ and positive integers r, k . Then, model checking ϕ over r -reversal bounded k -counter systems is solvable in PH.*

The same result holds in the presence of finitely many discrete clocks, where clock comparison constants are represented in unary. To prove this theorem, we first apply Proposition 7.5.2 and Lemma 6.2.4 to obtain the following corollary.

Proposition 7.5.5 *The reachability relation of a given n -state r -reversal bounded k -counter systems \mathcal{M} can be represented by an existential Presburger formula with $O(rk)$ quantifiers. Furthermore, this can be computed in time polynomial in n , and exponential in r, k .*

In particular, this algorithm runs in polynomial time for fixed values of r and k . Observe also that the number of quantifiers is $O(rk)$ (i.e. independent on the number of states in \mathcal{M}) since each of the conjunctive queries in the union of conjunctive queries that we obtain from Proposition 7.5.2 has $O(rk)$ quantifiers and that, if \bar{y} and \bar{z} contain the same number of variables, we can use the standard logical equivalence

$$\exists \bar{y} \phi(\bar{x}, \bar{y}) \vee \exists \bar{z} \phi'(\bar{x}, \bar{z}) \equiv \exists \bar{y} (\phi(\bar{x}, \bar{y}) \vee \phi'(\bar{x}, \bar{y})).$$

To complete the proof of Theorem 7.5.4, we shall recall the following result by Grädel [Grä88].

Proposition 7.5.6 ([Grä88]) *Fix a number $d > 0$. Checking whether a Presburger formula ϕ in prenex-normal form with at most d quantifiers can be done in PH. Furthermore, this still holds in the presence of linear (in)equations and numeric constants represented in binary.*

We are now ready to prove Theorem 7.5.4. The goal is to reduce the original EF model checking problem into evaluating Presburger formulas in prenex-normal form with a fixed number of quantifiers. This can be achieved as follows. Convert the fixed EF-formula ϕ into an equivalent FO(Reach) formula $\phi'(x)$ in prenex normal form. We then translate each formula of the form $\text{Reach}_\Gamma(y, z)$ into an existential Presburger formula with at most $O(rk)$ quantifiers using Proposition 7.5.5. In doing

so, we also replace each variable in φ' with a tuple of $O(rk)$ variables (recall that each configuration consists of counter values). Again, we may move all the quantifiers outside resulting in a formula with at most $O(rk\|\varphi'\|)$ quantifiers (i.e. a fixed number). Formulas of the form \rightarrow_a can be translated into a quantifier-free Presburger formula. Altogether, we obtain a Presburger formula $\psi(\bar{x})$ with a fixed number of quantifiers in polynomial time to which our original problem is reduced. Theorem 7.5.4 is then immediate from Grädel's result above.

Remark 7.5.1 It is possible to give a matching PH lower bound for this problem in the sense that, for each $n \in \mathbb{N}$, there exists two positive integers r and k , and an EF formula φ such that the problem of model checking φ over r -reversal k -counter systems is hard for Σ_n^P . This can be easily shown using the lower bound technique in Chapter 9. ■

Finally, what about the problem of model checking CTL over reversal-bounded counter systems? It turns out that this problem is undecidable. In fact, this already holds for 1-reversal 3-counter systems and very simple CTL formulas.

Proposition 7.5.7 *Model checking CTL over 1-reversal 3-counter systems is undecidable.*

This can be proven by a reduction from the undecidability problem of checking emptiness of languages of deterministic 0-reversal 3-counter systems that *may test equality of the current values of two counters* [ISD⁺02]. The proof can be found in the appendix.

Chapter 8

One-counter processes

In this chapter, we shall study another decidable restriction of Minsky's counter machines. It is well-known that two-counter machines are sufficient to obtain Turing-powerful models of computation [Min67]. On the other hand, this is not the case when we restrict the number of counters to one since such machines can now be viewed as a special case of pushdown systems with just one stack symbol, plus a non-removable bottom symbol which indicates an empty stack (and thus allows to test the counter for zero). Such machines are often referred to as *one-counter processes (OCPs)*. The aim of this chapter is to obtain a precise computational complexity of the EF-logic model checking problem over one-counter processes and show how this can be used to derive an optimal complexity for the problem of *weak-bisimilarity checking of one-counter processes against finite systems*.

Recall that, although our generic approach is able to derive decidability of EF-logic over PDSs (and hence OCPs), Theorem 5.5.1 shows that this approach will not provide an elementary upper bound for the problem. In contrast, the precise complexity of EF-logic model checking over PDSs is only PSPACE-complete [BEM97, Wal00]. The PSPACE lower bound was first proven by Bouajjani, Esparza and Maler [BEM97] even for a fixed EF formula, and only later the PSPACE upper bound was provided by Walukiewicz [Wal00]. This immediately gives a PSPACE upper bound for the combined complexity of EF-logic model checking over OCPs. On the other hand, the best lower bound known for this problem was only DP-hard [JKMS04], which is below the second level of the polynomial hierarchy.

The first hint that the computational complexity of model checking over OCPs could be easier than the same problem considered over PDSs is Serre's result [Ser06] that μ -calculus model checking over OCPs is PSPACE-complete. This is lower than

the EXP complexity of the same problem over PDSs due to Walukiewicz [Wal01]. In the case of equivalence checking, it is known that strong-bisimilarity checking against finite systems is solvable in polynomial time for OCPs [Kuc00], which is lower than the PSPACE complexity of the same problem when considered over PDSs [KM02, May00].

In this chapter, we shall present the recent result of Göller, Mayr, and the author [GMT09] that the problem of model checking EF-logic over OCPs is in P^{NP} , which is below the second level of the polynomial hierarchy. They also showed that P^{NP} -hardness can be obtained when the formula is represented as a dag, although this lower bound has recently been superseded by the recent Göller and Lohrey's P^{NP} lower bound [GL10] for the standard representation of EF formulas. The P^{NP} upper bound is derived by establishing a close correspondence of the problem with the membership problem of a fragment of Presburger arithmetic, which we call *Min-Max Arithmetic (MMA)*, which we also show to be P^{NP} -complete. We shall present this Min-Max Arithmetic in Section 8.2 and prove that it has a P^{NP} membership problem. In Section 8.3, we show how to efficiently transform OCPs into a suitable normal form. In Section 8.4, we provide a polynomial-time translation from the model checking problems of OCPs in this normal form to the membership problem of MMA. Combining this translation with the P^{NP} upper bound for the membership problem of MMA, we immediately obtain a P^{NP} upper bound for EF model checking over OCPs.

One of the most interesting applications of the P^{NP} upper bound for the problem of EF-logic model checking over OCPs is an immediate application to the problem of weak-bisimilarity checking of OCPs against finite systems. In Section 8.5, we show how a P^{NP} upper bound for the latter can be derived. We shall show in Section 8.6 a matching P^{NP} lower bound for this problem. This complexity is lower than the complexity for the corresponding problem for PDSs, which is PSPACE-complete [KM02, May00]. Finally, we shall conclude Section 8.6 with $P^{NP[\log]}$ lower bounds for weak-bisimilarity checking of OCPs against a *fixed* finite system, and for model checking a *fixed* EF formula against OCPs.

8.1 Preliminaries

In this section, we shall fix some notations that we will use throughout this chapter.

One-counter processes

We shall first use a more convenient notation for one-counter processes.

Definition 8.1.1 A one-counter process (OCP) is a tuple $\mathcal{P} = (Q, \delta_0, \delta_{>0})$, where Q is a finite set of states (a.k.a. control locations), $\delta_0 \subseteq Q \times \text{ACT} \times Q \times \{0, 1\}$ is a finite set of zero transitions, and $\delta_{>0} \subseteq Q \times \text{ACT} \times Q \times \{-1, 0, 1\}$ is a finite set of positive transitions. The size of a one-counter process is defined as $\|\mathcal{P}\| = |Q| + |\delta_0| + |\delta_{>0}|$. A one-counter net is a one-counter process that additionally satisfies $\delta_0 \subseteq \delta_{>0}$.

An OCP $\mathcal{P} = (Q, \delta_0, \delta_{>0})$ defines a transition system $\mathfrak{S}_{\mathcal{P}} = \langle S, \{\rightarrow_a\}_{a \in \text{ACT}} \rangle$, where $S = Q \times \mathbb{N}$ and $(q, n) \rightarrow_a (q', n+k)$ if and only if either $n = 0$ and $(q, a, q', k) \in \delta_0$, or $n > 0$ and $(q, a, q', k) \in \delta_{>0}$. If $(q_1, n_1) \rightarrow_a (q_2, n_2)$ for some $a \in \text{ACT}$, we also write $(q_1, n_1) \rightarrow_{\mathcal{P}} (q_2, n_2)$ (or even $(q_1, n_1) \rightarrow (q_2, n_2)$ if \mathcal{P} is understood). Notice that this definition coincides with the definition of 1-counter systems from Example 3.1.4.

EF dag-formulas

We now formalize the intuitive notion of EF formulas represented as directed acyclic graphs (dag). An EF *dag-formula* over ACT is a finite sequence of definitions $\varphi = (\varphi_i)_{i \in [l]}$ for some $l \in \mathbb{N}$, where for each $i \in [l]$ the *definition* φ_i is exactly one of the following, either:

1. $\varphi_i = \top$,
2. $\varphi_i = \neg \varphi_j$ for some $j \in [i-1]$,
3. $\varphi_i = \varphi_j \wedge \varphi_k$ for some $j, k \in [i-1]$,
4. $\varphi_i = \langle a \rangle \varphi_j$ for some $a \in \text{ACT}$ and some $j \in [i-1]$, or
5. $\varphi_i = \text{EF}_{\Gamma} \varphi_j$ for some $j \in [i-1]$ and $\Gamma \subseteq \text{ACT}$.

For each $i \in [l]$, we define the *size* $\|\varphi_i\| = 1$ if $\varphi_i = \top$ and $\|\varphi_i\| = \lceil \log i \rceil$ otherwise. The *size* of φ is defined as $\|\varphi\| = \sum_{i=1}^l \|\varphi_i\|$. Define the partial order $\prec_{\varphi} \subseteq [l] \times [l]$ as $(j, i) \in \prec_{\varphi}$ if and only if φ_j appears in the definition of φ_i . If $j \prec_{\varphi}^+ i$, we say that φ_j is a *subformula* of φ_i . Note that i is minimal with respect to \prec_{φ}^+ whenever $\varphi_i = \top$. Observe that $([l], \prec_{\varphi})$ is a dag. The standard definition of EF formulas coincide with this definition when $([l], \prec_{\varphi})$ is a directed tree. In this chapter, we shall call them EF *tree-formulas* to emphasize this fact. Next, we define the semantics. For this, let

$\mathfrak{S} = \langle S, \{\rightarrow_a\}_{a \in \text{ACT}} \rangle$ be a transition system. Let us define $\llbracket \varphi_i \rrbracket_{\mathfrak{S}} \subseteq S$ for each $i \in [l]$ by induction on \prec_{φ}^+ as follows:

1. $\llbracket \top \rrbracket_{\mathfrak{S}} = S$,
2. $\llbracket \neg \varphi_j \rrbracket_{\mathfrak{S}} = S \setminus \llbracket \varphi_j \rrbracket_{\mathfrak{S}}$,
3. $\llbracket \varphi_j \wedge \varphi_k \rrbracket_{\mathfrak{S}} = \llbracket \varphi_j \rrbracket_{\mathfrak{S}} \cap \llbracket \varphi_k \rrbracket_{\mathfrak{S}}$,
4. $\llbracket \langle a \rangle \varphi_j \rrbracket_{\mathfrak{S}} = \{s \in \mathfrak{S} \mid \exists t \in \llbracket \varphi_j \rrbracket_{\mathfrak{S}} : s \rightarrow_a t\}$, and
5. $\llbracket \text{EF}_{\Gamma} \varphi_j \rrbracket_{\mathfrak{S}} = \{s \in S \mid \exists t \in \llbracket \varphi_j \rrbracket_{\mathfrak{S}} : s \rightarrow_{\Gamma}^* t\}$.

We define $\llbracket \varphi \rrbracket_{\mathfrak{S}} = \llbracket \varphi_l \rrbracket_{\mathfrak{S}}$. We also write $\mathfrak{S}, s \models \varphi_i$ whenever $s \in \llbracket \varphi_i \rrbracket_{\mathfrak{S}}$. We deal with the *model checking problem for EF-logic over one-counter processes* defined as follows:

MODEL CHECKING EF-LOGIC OVER OCPs

Instance: An OCP \mathcal{P} , a configuration (q, n) of $\mathfrak{S}_{\mathcal{P}}$ with n in binary, and an EF dag/tree-formula φ .

Question: $\mathfrak{S}_{\mathcal{P}}, (q, n) \models \varphi$?

.

8.2 Min-Max Arithmetic

In this section we introduce a suitable representation of natural numbers in terms of a logic that we call MMA for *Min-Max Arithmetic*. In fact, the sets definable in MMA equal the sets definable in Presburger Arithmetic with one free variable or equivalently the one-dimensional semilinear sets. MMA can be seen as a syntactic variant of Presburger Arithmetic that is tailored towards having a fairly low complexity (P^{NP}) of the membership problem.

8.2.1 The definition

Formally, an MMA *dag-formula* is a sequence of definitions $\alpha = (\alpha_i)_{i \in [l]}$ for some $l \geq 1$, where for each $i \in [l]$ the *definition* α_i is precisely one of the following, where $j, k \in [i-1]$ and where $\sim \in \{\leq, \geq\}$:

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

2. $\sim n$, where $n \in \mathbb{N}$,
3. $\neg \alpha_j$,
4. $\alpha_j \wedge \alpha_k$,
5. $\sim \min \alpha_j$,
6. $n \sim \min \alpha_j$, where $n \in \mathbb{N}$,
7. $\sim \max(\alpha_j, n)$, where $n \in \mathbb{N}$, or
8. $m \sim \max(\alpha_j, n)$, where $m, n \in \mathbb{N}$.

We call α_i *atomic* in case it is of type (1) or (2). We will introduce usual abbreviations $<$, $>$, and $=$ with expected meanings. We formally put $\min \emptyset = \infty$, $\max \emptyset = -1$. Moreover we put $k \leq \infty$ and $k \not\leq \infty$, $k \not\leq -1$, and $k \geq -1$ for each $k \in \mathbb{N}$. Define the binary relation $\prec_\alpha \subseteq [l] \times [l]$ as $j \prec_\alpha i$ if and only if α_j occurs in the definition of α_i for each $i, j \in [l]$. Note that $([l], \prec_\alpha)$ is a dag and hence $([l], \prec_\alpha^+)$ is a strict partial order. Recall that i is minimal with respect to \prec_α^+ if and only if α_i is atomic. We say α is a *MMA tree-formula* if $([l], \prec_\alpha)$ is a directed tree. Let us now define the semantics of MMA dag-formulas. For each α_i we define the set $\llbracket \alpha_i \rrbracket \subseteq \mathbb{N}$ by induction on i w.r.t. \prec_α^+ as follows:

1. $\llbracket \equiv m \bmod n \rrbracket = \{k \in \mathbb{N} \mid k \equiv m \bmod n\}$,
2. $\llbracket \sim n \rrbracket = \{k \in \mathbb{N} \mid k \sim n\}$,
3. $\llbracket \neg \alpha_j \rrbracket = \mathbb{N} \setminus \llbracket \alpha_j \rrbracket$,
4. $\llbracket \alpha_j \wedge \alpha_k \rrbracket = \llbracket \alpha_j \rrbracket \cap \llbracket \alpha_k \rrbracket$,
5. $\llbracket \sim \min \alpha_j \rrbracket = \{k \in \mathbb{N} \mid k \sim \min \llbracket \alpha_j \rrbracket\}$,
6. $\llbracket n \sim \min \alpha_j \rrbracket = \begin{cases} \mathbb{N} & \text{if } n \sim \min \llbracket \alpha_j \rrbracket \\ \emptyset & \text{otherwise} \end{cases}$,
7. $\llbracket \sim \max(\alpha_j, n) \rrbracket = \{k \in \mathbb{N} \mid k \sim \max(\llbracket \alpha_j \rrbracket \cap [0, n])\}$,
- 8.

$$\llbracket m \sim \max(\alpha_j, n) \rrbracket = \begin{cases} \mathbb{N} & \text{if } m \sim \max(\llbracket \alpha_j \rrbracket \cap [0, n]) \\ \emptyset & \text{otherwise.} \end{cases}$$

Observe that MMA can be seen as a fragment of Presburger Arithmetic. We define $\llbracket \alpha \rrbracket = \llbracket \alpha_i \rrbracket$. We call α *valid* if $\llbracket \alpha \rrbracket = \mathbb{N}$, for example ≥ 0 is valid. Define the *size* $\|\alpha_i\|$ by case distinction as follows: $\|\equiv m \bmod n\| = \|\sim n\| = \lceil \log n \rceil$, $\|\neg \alpha_j\| = \lceil \log j \rceil$, $\|\alpha_j \wedge \alpha_k\| = \lceil \log j \rceil + \lceil \log k \rceil$, $\|\sim \min \alpha_j\| = \lceil \log j \rceil$, $\|n \sim \min \alpha_j\| = \lceil \log n \rceil + \lceil \log j \rceil$, $\|\sim \max(\alpha_j, n)\| = \lceil \log j \rceil + \lceil \log n \rceil$, and finally $\|m \sim \max(\alpha_j, n)\| = \lceil \log m \rceil + \lceil \log j \rceil + \lceil \log n \rceil$. Define the *size* of α as $\|\alpha\| = \sum_{i \in [l]} \|\alpha_i\|$. For better readability, we will allow more complex definitions such as e.g. $\alpha_i = \neg \alpha_j \wedge (x \equiv 3 \bmod 5)$.

8.2.2 Syntactic sugar: extended MMA

In order to ease our reduction from model checking OCP to evaluating MMA formulas, we introduce *extended MMA formulas*. Extended MMA formulas allow definitions of the kind $\alpha_i = \alpha_j - 1$ and $\alpha_i = \alpha_j + 1$. For $\odot \in \{+, -\}$, the semantics is defined as $\llbracket \alpha_j \odot 1 \rrbracket = (\llbracket \alpha_j \rrbracket \odot 1) \cap \mathbb{N}$. Observe that, in general, the two operators cannot be interchanged, i.e. we generally do *not* have $\llbracket (\alpha - 1) + 1 \rrbracket = \llbracket (\alpha + 1) - 1 \rrbracket$. On the other hand, observe that $\llbracket \alpha \rrbracket = \llbracket (\alpha + 1) - 1 \rrbracket$. Define the *size* of $\|\alpha_j + 1\| = \|\alpha_j - 1\| = \lceil \log j \rceil + 1$. For each natural k define $\alpha_j \odot k$ to be the abbreviation for

$$\underbrace{(\cdots (\alpha_j \odot 1) \cdots)}_{k \text{ many times}} \odot 1$$

for each $\odot \in \{-, +\}$.

8.2.3 Basic properties of MMA and extended MMA

We shall now prove basic properties that are satisfied by MMA and extended MMA dag-formulas. We shall first prove a periodicity lemma for extended MMA dag-formulas, i.e., sufficiently large numbers which are satisfied by extended MMA dag-formulas have nice periodic behaviors. We shall also show that extended MMA are neither more expressive nor more succinct by providing a polynomial-time translation from extended MMA dag-formulas to equivalent MMA dag-formulas. Using these results, we shall show that the membership problem for extended MMA dag-formulas can be solved in P^{NP} .

We shall start with a simple but important periodicity lemma for extended MMA dag-formulas. Let $\alpha = (\alpha_i)_{i \in [l]}$ be an extended MMA dag-formula. Define v_i to be maximal number n such that α_j is $\sim n$ or $\sim \max(\alpha_j, n)$ for some $j \in [i]$. Define L_i to be the least common multiple of all $n > 0$ such that the definition of α_j is $\equiv m \bmod n$

for some $j \in [i-1]$ and some $m \in \mathbb{Z}/n\mathbb{Z}$, and 1 if no such formula exists. Observe that $i < j$ implies $L_i | L_j$ and moreover $L_i \in \exp(\|\alpha\|)$, thus polynomially (in $\|\alpha\|$) many bits suffice to represent each L_i .

Lemma 8.2.1 (Periodicity Lemma for extended MMA) *Let $i \in [l]$ and assume that $n_1, n_2 > i \cdot L_i + v_i$. Then the following implication holds:*

$$n_1 \equiv n_2 \pmod{L_i} \quad \Rightarrow \quad (n_1 \in \llbracket \alpha_i \rrbracket \Leftrightarrow n_2 \in \llbracket \alpha_i \rrbracket)$$

Proof. We prove the lemma by induction on \prec_α^+ . For the induction base, assume that i is minimal with respect to \prec_α^+ , i.e. α_i is atomic.

Case α_i is $\equiv m \pmod n$, for some $n > 0$ and some $m \in \mathbb{Z}/n\mathbb{Z}$. Observe that the implication holds since L_i is a multiple of n by definition.

Case α_i is $\sim n$, where $\sim \in \{\leq, \geq\}$ and where $n \in \mathbb{N}$. By definition we have $v_i \geq n$. The implication clearly holds since natural numbers exceeding n are either all contained in $\llbracket \alpha_i \rrbracket$ or are all not contained in $\llbracket \alpha_i \rrbracket$.

For the induction step, assume that i is not minimal with respect to \prec_α^+ . For this, we make a case distinction according to α_i .

Case $\alpha_i = \neg\alpha_j$ for some $j \in [i-1]$. The required implication holds trivially due to induction hypothesis and the fact that $j \cdot L_j + v_j \leq i \cdot L_i + v_i$ and $L_i = L_j$.

Case $\alpha_i = \alpha_j \wedge \alpha_k$ for some $j, k \in [i-1]$. First, we have that both L_k and L_j divide L_i . Second, both $j \cdot L_j + v_j$ and $k \cdot L_k + v_k$ are at most $i \cdot L_i + v_i$. Now let $n_1, n_2 > i \cdot L_i + v_i$ and assume $n_1 \equiv n_2 \pmod{L_i}$. Then we have $n_1 \in \llbracket \alpha_i \rrbracket$ if and only if $n_1 \in \llbracket \alpha_j \rrbracket$ and $n_1 \in \llbracket \alpha_k \rrbracket$. By induction hypothesis, the latter is equivalent to $n_2 \in \llbracket \alpha_j \rrbracket$ and $n_2 \in \llbracket \alpha_k \rrbracket$ which is in turn equivalent to $n_2 \in \llbracket \alpha_i \rrbracket$.

Case α_i is $\sim \min \alpha_j$ for some $\sim \in \{\leq, \geq\}$ and for some $j \in [i-1]$. We claim that the implication holds for i by distinguishing if $\llbracket \alpha_j \rrbracket$ is empty or not. In case $\llbracket \alpha_j \rrbracket = \emptyset$, (recall $\min \emptyset = \infty$), then $\llbracket \alpha_i \rrbracket$ either equals \mathbb{N} or \emptyset , depending on \sim . The implication obviously holds in this case. In case $\llbracket \alpha_j \rrbracket \neq \emptyset$, then there exists some $n \in \llbracket \alpha_j \rrbracket$ with $n \leq j \cdot L_j + v_j + L_j$ by induction hypothesis. Observe that the latter is less than or equal to $(i-1) \cdot L_i + v_i + L_i = i \cdot L_i + v_i$. Again, depending on \sim , all naturals exceeding n (in particular naturals exceeding $i \cdot L_i + v_i$) are either all in $\llbracket \alpha_i \rrbracket$ or are all not in $\llbracket \alpha_i \rrbracket$. Thus, the implication holds.

Case $\alpha_i = n \sim \min \alpha_j$ for some $n \in \mathbb{N}$, some $\sim \in \{\leq, \geq\}$, and some $j \in [i-1]$. Since $\llbracket \alpha_i \rrbracket$ either equals \mathbb{N} or \emptyset , the implication trivially holds.

Case α_i is $\sim \max(\alpha_j, n)$ for some $\sim \in \{\leq, \geq\}$, some $j \in [i-1]$, and some $n \in \mathbb{N}$. First observe that $n \leq v_i$ by definition. Second, all naturals exceeding n (in particular those exceeding v_i) either all satisfy α_i or all do not, depending on \sim . Thus, the implication holds.

Case α_i is $m \sim \max(\alpha_j, n)$ for some $\sim \in \{\leq, \geq\}$, some $j \in [i-1]$, and some $m, n \in \mathbb{N}$. Since $\llbracket \alpha_i \rrbracket$ either equals \mathbb{N} or \emptyset , the implication holds trivially.

Case $\alpha_i = \alpha_j \odot 1$ for some $j \in [i-1]$ and some $\odot \in \{+, -\}$. The implication follows directly from $j \cdot L_j + v_j + 1 \leq (i-1) \cdot L_i + v_i + L_i = i \cdot L_i + v_i$ and induction hypothesis.

□

It turns out that extended MMA formulas are neither more expressive nor more succinct than MMA formulas.

Lemma 8.2.2 *The following problem is computable in polynomial time:*

INPUT: An extended MMA dag-formula α .

OUTPUT: A MMA dag-formula β such that $\llbracket \alpha \rrbracket = \llbracket \beta \rrbracket$.

The proof of this lemma, which is not difficult but rather tedious, is given in the appendix. Now, by a bottom-up computation and combining Lemma 8.2.1 and Lemma 8.2.2, we can deduce a P^{NP} upper bound for the membership problem for extended MMA.

Proposition 8.2.3 *The following problem is in P^{NP} :*

INPUT: $n_0 \in \mathbb{N}$ in binary and an extended MMA dag-formula α .

QUESTION: $n_0 \in \llbracket \alpha \rrbracket$?

Proof. First, it is easy to see that if α is an MMA dag-formula with no occurrence of min and max then checking whether $n_0 \in \llbracket \alpha \rrbracket$ can be done in polynomial time. We shall make use of this basic fact in the proof. In a first step, we apply Lemma 8.2.2 and compute in polynomial time an MMA dag-formula formula β such that $\llbracket \beta \rrbracket = \llbracket \alpha \rrbracket$. Assume $\beta = (\beta_i)_{i \in [l]}$. In a second step, we eliminate min and max operators that occur in β inductively on \prec_β^+ .

For this, let $i \in [l]$ be minimal with respect to \prec_β^+ such that the definition β_i contains either min or max. In case β_i is $\sim \min \beta_j$, we know that for each $n_1, n_2 > j \cdot L_j + v_j$ with $n_1 \equiv n_2 \pmod{L_j}$ we have $n_1 \in \llbracket \beta_j \rrbracket$ if and only if $n_2 \in \llbracket \beta_j \rrbracket$ by Lemma 8.2.1. This implies that $\min \llbracket \beta_j \rrbracket \in [0, (j+1) \cdot L_j + v_j] \cup \{\infty\}$. Moreover, observe that $(j+1) \cdot L_j + v_j$ can be represented using polynomially many bits in $\|\beta\|$. Furthermore, note that we

can decide deterministically in polynomial time whether, for a given m given in binary, we have $m \in \llbracket \beta_j \rrbracket$, since neither the definition of β_j nor the definition of β_k contains min or max, for each $k \prec_{\beta}^+ j$. Via a binary search method, we can compute

$$\mu = \min\{m \in [0, (j+1) \cdot L_j + v_j] \mid m \in \llbracket \beta_j \rrbracket\}$$

by some deterministic polynomial time bounded Turing machine that has access to an NP oracle. After that, we “symbolically” modify β by replacing β_i ’s previous definition $\sim \min \beta_j$ by $\sim \mu$. The cases when $\beta_i = n \sim \min \beta_j$, $\beta_i = \sim \max(\beta_j, n)$, or $\beta_i = m \sim \max(\beta_j, n)$ can be dealt with analogously.

We repeat this replacement process until β does not contain any min or max operator. Finally, we check if $n_0 \in \llbracket \beta \rrbracket$ in polynomial time. \square

8.3 Saturations and small arithmetic progressions

For the rest of this section, let us fix some one-counter process $\mathcal{P} = (Q, \delta_0, \delta_{>0})$. For technical reasons, we add a new transition label $\lambda \in \text{ACT}$ 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 so that we only have to consider *normalized paths*, i.e. paths, where the sequence of counter values of the involved configurations are first non-increasing and then non-decreasing. Let \mathcal{P}' denote the resulting OCP *after saturation*. Our saturation construction has the following motivation: (1) We can compute in polynomial time all information needed for representing normalized paths in $\mathfrak{S}_{\mathcal{P}'}$ in terms of few “small” arithmetic progressions, and (2) for every EF dag-formula φ in which λ does not occur we have $\mathfrak{S}_{\mathcal{P},s} \models \varphi$ if and only if $\mathfrak{S}_{\mathcal{P}',s} \models \varphi$ for every configuration $s \in Q \times \mathbb{N}$.

8.3.1 Saturation construction

Given a nonempty path $\pi = (q_1, n_1) \rightarrow_{\mathcal{P}} (q_2, n_2) \cdots \rightarrow_{\mathcal{P}} (q_k, n_k)$ in $\mathfrak{S}_{\mathcal{P}}$, we call π *mountain*, if $n_1 = n_k$ and $n_i \geq n_1$ for each $i \in [k]$. We call π *zero*, if $n_i = 0$ for some $i \in [k]$, otherwise we call π *positive*. Let $(q_1, n_1), (q_2, n_2) \in Q \times \mathbb{N}$ be configurations. Then, we write $(q_1, n_1) \downarrow_{\mathcal{P}} (q_2, n_2)$ (resp. $(q_1, n_1) \uparrow_{\mathcal{P}} (q_2, n_2)$) whenever $(q_1, n_1) \rightarrow_{\mathcal{P}} (q_2, n_2)$ 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 locations $q, q' \in Q$ and $\delta \in \{\delta_0, \delta_{>0}\}$, we now present rules (R1) to (R4) that can be applied only if $(q, \lambda, q', 0) \notin \delta$. In this case, we can add the transition $(q, \lambda, q', 0)$ to δ if at least one of the following conditions holds:

- (R1) $(q, a, q', 0) \in \delta$ for some $a \in \text{ACT}$.
- (R2) $(q, a_1, q_1, +1) \in \delta$ and $(q_1, a_2, q', -1) \in \delta_{>0}$ for some $q_1 \in Q$ and some $a_1, a_2 \in \text{ACT}$.
- (R3) $(q, a_1, q_1, +1) \in \delta$, $(q_1, \lambda, q_2, 0) \in \delta_{>0}$, and $(q_2, a_2, q', -1) \in \delta_{>0}$ for some $q_1, q_2 \in Q$ and some $a_1, a_2 \in \text{ACT}$.
- (R4) $(q, \lambda, q_1, 0) \in \delta$ and $(q_1, \lambda, q', 0) \in \delta$ for some $q_1 \in Q$.

Formally, let $\mathcal{P}' = (Q, \delta'_0, \delta'_{>0})$ denote the unique one-counter process that we obtain from \mathcal{P} by applying rules (R1)–(R4) until it is no longer possible. The following lemma, whose proof is in the appendix, shows that reachability in this saturated OCP \mathcal{P}' can be witnessed by a path, in which the changes in the counter values is extremely simple: monotonically non-increasing and then monotonically non-decreasing.

Lemma 8.3.1 *Let $s, t \in Q \times \mathbb{N}$ be configuration. Then, the following three statements are equivalent:*

1. $s \rightarrow_{\mathcal{P}}^* t$.
2. $s \rightarrow_{\mathcal{P}'}^* t$.
3. *There exists some configuration $u \in Q \times \mathbb{N}$ such that $s \downarrow_{\mathcal{P}'}^* u$ and $u \uparrow_{\mathcal{P}'}^* t$.*

8.3.2 Computing small arithmetic progressions

Observe that if $(q_1, n_1) \uparrow_{\mathcal{P}'}^* (q_2, n_2)$ and $n_1 > 0$, then also $(q_1, n_1 + i) \uparrow_{\mathcal{P}'}^* (q_2, n_2 + i)$ for each $i \in \mathbb{N}$. Similarly, if $(q_1, n_1) \downarrow_{\mathcal{P}'}^* (q_2, n_2)$ and $n_2 > 0$, then also $(q_1, n_1 + i) \downarrow_{\mathcal{P}'}^* (q_2, n_2 + i)$ for each $i \in \mathbb{N}$. This motivates us to define, for each $q_1, q_2 \in Q$, the following sets of differences of counter values of monotone positive paths:

$$\begin{aligned} \Delta_{\uparrow}^{>0}(q_1, q_2) &= \{d \in \mathbb{N} \mid (q_1, 1) \uparrow_{\mathcal{P}'}^* (q_2, d+1)\} \\ \Delta_{\downarrow}^{>0}(q_1, q_2) &= \{d \in \mathbb{N} \mid (q_1, d+1) \downarrow_{\mathcal{P}'}^* (q_2, 1)\} \end{aligned}$$

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

$$\begin{aligned}\Delta_{\uparrow}^0(q_1, q_2) &= \{d \in \mathbb{N} \mid (q_1, 0) \uparrow_{\mathcal{P}'}^* (q_2, d)\} \\ \Delta_{\downarrow}^0(q_1, q_2) &= \{d \in \mathbb{N} \mid (q_1, d) \downarrow_{\mathcal{P}'}^* (q_2, 0)\}\end{aligned}$$

It turns out that these sets can be expressed as a union of a small number of arithmetic progressions whose offsets and periods are also small. Furthermore, the computation of these arithmetic progressions from \mathcal{P}' can be done efficiently.

Lemma 8.3.2 *Each of the sets $\Delta_{\uparrow}^0(q_1, q_2)$, $\Delta_{\downarrow}^0(q_1, q_2)$, $\Delta_{\uparrow}^{>0}(q_1, q_2)$, $\Delta_{\downarrow}^{>0}(q_1, q_2)$ can be represented as 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.*

This lemma can be proven by treating \mathcal{P}' as NWAs over a unary alphabet and applying the special case of Theorem 7.3.1 over a unary alphabet.

Proof. To obtain better polynomial upper bounds, we recall Chrobak-Martinez's Theorem [Chr86, Mar02, To09b], which is a previously known special case of Theorem 7.3.1 over unary alphabets: given an NWA \mathcal{A} over the alphabet $\{\#\}$ with n states, the set $\mathcal{P}((\mathcal{L}(\text{Aut})))$ of the lengths of words in $\mathcal{L}(\mathcal{A})$ can be represented as a union of $O(n^2)$ arithmetic progressions with offsets bounded by $O(n^2)$ and periods bounded by $O(n)$; moreover, this can be computed in polynomial time. We shall only show Lemma 8.3.2 for $\Delta_{\uparrow}^0(q_1, q_2)$; the other cases can be proven analogously. Let

$$\mathcal{A} = (\{\#\}, Q \times \{0, 1\}, \delta, q_1, \{(q_2, 0), (q_2, 1)\})$$

be the NWA (with ε transitions) such that

- $\delta((q, 0), \#) = \{(q', 1) : \exists a \in \text{ACT} : (q, a, q', 1) \in \delta'_0\},$
- $\delta((q, 1), \#) = \{(q', 1) : \exists a \in \text{ACT} : (q, a, q', 1) \in \delta'_{>0}\},$
- $\delta((q, 0), \varepsilon) = \{(q', 0) : \exists a \in \text{ACT} : (q, a, q', 0) \in \delta'_0\},$ and
- $\delta((q, 1), \varepsilon) = \{(q', 1) : \exists a \in \text{ACT} : (q, a, q', 0) \in \delta'_{>0}\}.$

That is, \mathcal{A} is obtained from \mathcal{P}' by regarding it as an NWA over $\{\#\}$ by removing all pop-transitions, treating zero transitions as ε -transitions, and treating push transitions as reading the symbol $\#$. Observe that \mathcal{A} simulates precisely paths in $\mathfrak{S}_{\mathcal{P}'}$ from $(q_1, 0)$

to some configuration in $\{q_2\} \times \mathbb{N}$ on which the counter behaves monotonically non-decreasing. It is easy to see now that $\mathcal{P}(\mathcal{L}(\mathcal{A}))$ coincides with $\Delta_{\uparrow}^=0(q_1, q_2)$. Since we may compute an equivalent NWA without ε -transitions and with the same number of states in polynomial time, Chrobak-Martinez's Theorem implies that we can compute from \mathcal{A} in polynomial time a union of $O(n^2)$ arithmetic progressions, with offsets bounded by $O(n^2)$ and periods bounded by $O(n)$, representing $\Delta_{\uparrow}^=0(q_1, q_2)$. \square

8.3.3 Characterization of zero paths and positive paths

Lemma 8.3.2 will now be used to obtain characterizations of zero paths and positive paths in $\mathfrak{S}'_{\mathcal{P}}$. Let $q_1, q_2 \in Q$ be control locations. Note that if $(q_1, n) \downarrow_{\mathcal{P}'}^* (q_3, 1) \uparrow_{\mathcal{P}'}^* (q_2, n)$ for some $q_3 \in Q$, then also $(q_1, n+i) \downarrow_{\mathcal{P}'}^* (q_3, 1+i) \uparrow_{\mathcal{P}'}^* (q_2, n+i)$. Therefore, we define $\nabla(q_1, q_2) \in \mathbb{N} \cup \{\infty\}$ to be

$$\min\{n > 0 \mid \exists q_3 \in Q : (q_1, n) \downarrow_{\mathcal{P}'}^* (q_3, 1) \uparrow_{\mathcal{P}'}^* (q_2, n)\}.$$

Observe that $\nabla(q, q) = 1$ for every $q \in Q$. The following lemma, whose proof is in the appendix, can be easily proven using Lemma 8.3.2 as a subroutine.

Lemma 8.3.3 *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.*

We now obtain a characterization for zero paths.

Lemma 8.3.4 *There is a zero path from (q, n) to (q', n') in $\mathfrak{S}_{\mathcal{P}'}$ if and only if $n \in \Delta_{\downarrow}^=0(q, q'')$ and $n' \in \Delta_{\uparrow}^=0(q'', q')$ for some $q'' \in Q$.*

Proof. “If”: Follows directly from definition of the sets $\Delta_{\downarrow}^=0$ and $\Delta_{\uparrow}^=0$.

“Only-if”: Let $\pi = (q_1, n_1) \rightarrow_{\mathcal{P}'} (q_2, n_2) \cdots \rightarrow_{\mathcal{P}'} (q_k, n_k)$ be a zero path from (q, n) to (q', n') in $\mathfrak{S}_{\mathcal{P}'}$. Let $l = \min\{i \in [k] \mid n_i = 0\}$ and let $r = \max\{i \in [k] \mid n_i = 0\}$. Then observe that since $(q_1, n_1) \rightarrow_{\mathcal{P}'}^* (q_l, 0)$, it follows $(q_1, n_1) \downarrow_{\mathcal{P}'}^* (q_l, 0)$ due to the equivalences of points (2) and (3) of Lemma 8.3.1 and the definition of $\downarrow_{\mathcal{P}'}$. Analogously, we have $(q_r, 0) \uparrow_{\mathcal{P}'}^* (q_k, n_k)$. Moreover, there exists a (mountain) subpath ρ from $(q_l, 0)$ to $(q_r, 0)$ in π . By again applying the equivalences of points (2) and (3) of Lemma 8.3.1 the subpath ρ can be replaced by a path on which all states have counter value 0. Hence, altogether there exists some $q'' \in Q$ such that we have

$$(q_1, n_1) \downarrow_{\mathcal{P}}^* (q_l, 0) \downarrow_{\mathcal{P}'}^* (q'', 0) \uparrow_{\mathcal{P}'}^* (q_r, 0) \uparrow_{\mathcal{P}'}^* (q_k, n_k).$$

Hence we obtain $n_1 \in \Delta_{\downarrow}^=0(q, q'')$ and $n_k \in \Delta_{\uparrow}^=0(q'', q')$ as required. \square

The next lemma characterizes positive paths.

Lemma 8.3.5 *Assume $n \leq n'$. Then there exists a positive path in $\mathfrak{S}_{\mathcal{P}'}$ from (q, n) to (q', n') if and only if $n \geq \nabla(q, q'')$ and $n' - n \in \Delta_{\uparrow}^{>0}(q'', q')$ for some $q'' \in Q$.*

Assume $n \geq n'$. Then there exists a positive path from (q, n) to (q', n') in $\mathfrak{S}_{\mathcal{P}'}$ if and only if $n' \geq \nabla(q'', q')$ and $n - n' \in \Delta_{\downarrow}^{>0}(q, q'')$ for some $q'' \in Q$.

Proof. We only prove the case when $n \leq n'$. The case when $n \geq n'$ can be proven analogously.

“If”: Assume $n \geq \nabla(q, q'')$ and $n' - n \in \Delta_{\uparrow}^{>0}(q'', q')$ for some $q'' \in Q$. Then $(q, n) \downarrow_{\mathcal{P}'}^* (q_0, n - d) \uparrow_{\mathcal{P}'}^* (q'', n)$ for some $q_0 \in Q$ and some $d \in [0, n - 1]$ following immediately from definition of $\nabla(q, q'')$. Moreover $n' - n \in \Delta_{\uparrow}^{>0}(q'', q')$ implies $(q'', n) \uparrow_{\mathcal{P}'}^* (q', n')$. Altogether, there exists a positive path of the kind $(q, n) \downarrow_{\mathcal{P}'}^* (q_0, n - d) \uparrow_{\mathcal{P}'}^* (q', n')$ in $\mathfrak{S}_{\mathcal{P}'}$ as required.

“Only-if”: Let $\pi = (q_1, n_1) \rightarrow_{\mathcal{P}'}^* (q_2, n_2) \cdots (q_k, n_k)$ be a positive path from (q, n) to (q', n') . Let $\mu = \min\{n_i \mid i \in [k]\}$ be the minimal counter value that appears in π . Recall that $\mu > 0$ since π is positive. Define $l = \min\{i \in [k] \mid n_i = \mu\}$ and $r = \max\{i \in [k] \mid n_i = \mu\}$. Since $(q_1, n_1) \rightarrow_{\mathcal{P}'}^* (q_l, \mu)$, it follows from the equivalences of points (2) and (3) of Lemma 8.3.1 and the definition of $\downarrow_{\mathcal{P}'}$ that $(q_1, n_1) \downarrow_{\mathcal{P}'}^* (q_l, \mu)$. Analogously, it follows that $(q_r, \mu) \uparrow_{\mathcal{P}'}^* (q_k, n_k)$. Moreover the subpath from (q_l, μ) to (q_r, μ) in π can be replaced by a path of the kind $(q_l, \mu) \downarrow_{\mathcal{P}'}^* (q_\mu, \mu) \uparrow_{\mathcal{P}'}^* (q_r, \mu)$ for some $q_\mu \in Q$ by Lemma 8.3.1. Altogether, we have shown the existence of a path of the following kind

$$(q, n) \downarrow_{\mathcal{P}'}^* (q_\mu, \mu) \uparrow_{\mathcal{P}'}^* (q', n').$$

Thus, there exists some $q'' \in Q$ such that the previous path can be split up as

$$(q, n) \downarrow_{\mathcal{P}'}^* (q_\mu, \mu) \uparrow_{\mathcal{P}'}^* (q'', n) \uparrow_{\mathcal{P}'}^* (q', n').$$

This implies $n \geq \nabla(q, q'')$ and $n' - n \in \Delta_{\uparrow}(q'', q')$ as required. \square

8.4 A translation to MMA

In this section, we shall present a polynomial-time translation from the problem of model checking EF dag-formulas over OCPs to the membership problem of MMA. In fact, the following theorem shows a stronger statement: the set of configurations of an OCP satisfying the given EF dag-formula can be represented by MMA formulas; moreover, they can be computed in polynomial time.

Theorem 8.4.1 *From a given one-counter process \mathcal{P} and a given EF dag-formula φ , we can compute in polynomial time for each control location q of \mathcal{P} an MMA dag-formula $\alpha(q)$ such that $\llbracket \alpha(q) \rrbracket = \{n \in \mathbb{N} \mid (\mathfrak{S}_{\mathcal{P}}, (q, n)) \models \varphi\}$.*

By combining Theorem 8.4.1 with Proposition 8.2.3, the following corollary is immediate.

Corollary 8.4.2 *The problem of model checking EF dag-formulas over OCPs is in P^{NP} .*

Therefore, it remains to prove Theorem 8.4.1. For the rest of this section, let us fix an OCP $\mathcal{P} = (Q, \delta_0, \delta_{>0})$ and an EF dag-formula $\varphi = (\varphi_i)_{i \in [l]}$. For convenience, we shall assume that each occurrence of the EF_{Γ} operator in φ satisfies $\Gamma = \text{ACT}$ (or equivalently $\text{EF}_{\Gamma} = \text{EF}$); for the general case of $\Gamma \subseteq \text{ACT}$, the proof can easily be adapted by first restricting ourselves to Γ -transitions in \mathcal{P} when we see EF_{Γ} operators. Assume now that $Q = \{q_1, \dots, q_k\}$. For technical convenience, we will identify each element $(q_i, j) \in Q \times [l]$ with the corresponding natural number $i + (j - 1) \cdot k \in [k \cdot l]$. The goal of this section is to present a polynomial time algorithm to compute an extended MMA formula $\alpha = (\alpha_{(q,j)})_{(q,j) \in Q \times [l]}$ such that $\llbracket \alpha_{(q,j)} \rrbracket = \{n \in \mathbb{N} \mid (\mathfrak{S}_{\mathcal{P}}, (q, n)) \models \varphi_j\}$ for each $(q, j) \in Q \times [l]$.

First, we saturate \mathcal{P} in polynomial time. Then, we apply Lemma 8.3.2 and compute in polynomial time the sets $\Delta_{\downarrow}^{>0}(q, q')$, $\Delta_{\uparrow}^{>0}(q, q')$, $\Delta_{\downarrow}^{=0}(q, q')$, and $\Delta_{\uparrow}^{=0}(q, q')$, which are each unions of $O(|Q|^2)$ arithmetic progressions with offsets bounded by $O(|Q|^2)$ and periods bounded by $O(|Q|)$ for each $q, q' \in Q$. By applying Lemma 8.3.5 we compute in polynomial time $\nabla(q, q') \in [n_{\nabla}] \cup \{\infty\}$ for each $q, q' \in Q$, where $n_{\nabla} \in O(|Q|^2)$.

Let us now present the computation of the MMA formula $\alpha = (\alpha_{(q,j)})_{(q,j) \in Q \times [l]}$. We will do this by induction on j with respect to \prec_{φ}^+ and simultaneously for each $q \in Q$.

Base Case. Assume j is minimal with respect to \prec_{φ}^+ . Then $\varphi_j = \top$ and we put $\alpha_{(q,j)} = (\geq 0)$ for each $q \in Q$.

Induction Step.

Assume $\varphi_j = \neg \varphi_{j'}$ for some $j' \in [j - 1]$. Then we put $\alpha_{(q,j)} = \neg \alpha_{(q,j')}$ for each $q \in Q$.

Assume $\varphi_j = \varphi_{j_1} \wedge \varphi_{j_2}$ for some $j_1, j_2 \in [j - 1]$. Then we put $\alpha_{(q,j)} = \alpha_{(q,j_1)} \wedge \alpha_{(q,j_2)}$ for each $q \in Q$.

Assume $\varphi_j = \langle a \rangle \varphi_{j'}$ for some $a \in \text{ACT}$ and some $j' \in [j-1]$. By induction hypothesis, we have

$$\alpha_{(q',j')} = \{n \in \mathbb{N} \mid (q', n) \models \varphi_{j'}\}$$

for each $q' \in Q$. By putting $\hat{+} = -$ and $\hat{-} = +$, we define $\alpha_{(q,j)}$ as the conjunction of

$$(< 0) \rightarrow \left(\bigvee_{\substack{q' \in Q: \\ (q,a,q',+1) \in \delta_0}} \alpha_{(q',j')} - 1 \vee \bigvee_{\substack{q' \in Q: \\ (q,a,q',0) \in \delta_0}} \alpha_{(q',j')} \right)$$

and

$$(> 0) \rightarrow \left(\bigvee_{\substack{q' \in Q, \odot \in \{-,+\}: \\ (q,a,q',\odot 1) \in \delta_{>0}}} \alpha_{(q',j')} \hat{\odot} 1 \vee \bigvee_{\substack{q' \in Q: \\ (q,a,q',0) \in \delta_{>0}}} \alpha_{(q',j')} \right).$$

Finally, assume $\varphi_j = \text{EF} \varphi_{j'}$ for some $j' \in [j-1]$. Let us first fix control locations $q, q' \in Q$. By induction hypothesis

$$\llbracket \alpha_{(q',j')} \rrbracket = \{n \in \mathbb{N} \mid (q', n) \models \varphi_{j'}\}.$$

By Lemma 8.2.1 we know that for each $n_1, n_2 \in \mathbb{N}$ which exceed a threshold $t_{(q',j')}$ such that $n_1 \equiv n_2 \pmod{L_{(q',j')}}$ we have $n_1 \in \llbracket \alpha_{(q',j')} \rrbracket$ if and only if $n_2 \in \llbracket \alpha_{(q',j')} \rrbracket$. Note that this implies that $\llbracket \alpha_{(q',j')} \rrbracket$ is infinite if and only if there exists some $n \in \llbracket \alpha_{(q',j')} \rrbracket$ such that $t_{(q',j')} < n \leq t_{(q',j')} + L_{(q',j')}$.

Let us now fix an arithmetic progression $a + b\mathbb{N}$ with $b > 0$ that is a subset of some of the sets $\Delta_{\downarrow}^{>0}(q, q'), \Delta_{\uparrow}^{>0}(q, q'), \Delta_{\downarrow}^{=0}(q, q')$, or $\Delta_{\uparrow}^{=0}(q, q')$. Moreover, let $c \in \mathbb{Z}/b\mathbb{Z}$ be some residue class. We aim at defining an MMA formula $\text{inf}(q', j', c, b)$ that is valid if and only if there are infinitely many naturals that satisfy $\alpha_{(q',j')}$ and that are congruent c modulo b . Now observe that the latter is the case exactly whenever there exists some $n \in \llbracket \alpha_{(q',j')} \rrbracket$ such that $t_{(q',j')} < n \leq t_{(q',j')} + L_{(q',j')}$ and moreover $n \equiv c \pmod{\gcd(b, L_{(q',j')})}$. Let us define the auxiliary MMA formula

$$\psi(q', j', c, b) = \alpha_{(q',j')} \wedge (\equiv c \pmod{\gcd(b, L_{(q',j')})})$$

and finally

$$\text{inf}(q', j', c, b) = t_{(q',j')} < \max(\psi(q', j', b, c), t_{(q',j')} + L_{(q',j')}).$$

Next, we aim at defining the set

$$\{n \in \mathbb{N} \mid \exists n' \in \mathbb{N} : (q, n) \xrightarrow{*}_{\mathcal{P}} (q', n'), (q', n') \models \varphi_{j'}\}$$

in terms of an extended MMA formula. For this, assume that there is a path π from (q, n) to (q', n') in $\mathfrak{S}_{\mathcal{P}}$ such that $(q', n') \models \phi_{j'}$, where $n' \in \mathbb{N}$. We distinguish three (not necessarily distinct) cases. Either (1) π is positive and $n \leq n'$, (2) π is positive and $n \geq n'$, or (3) π is zero. We will realize each of these cases by corresponding extended MMA dag-formulas $\beta_1(q, q')$, $\beta_2(q, q')$, and $\beta_3(q, q')$ respectively.

Let us first consider case (1), i.e. π is positive and $n \leq n'$. Then $n \geq \nabla(q, q'')$ and $n' - n \in \Delta_{\uparrow}^{>0}(q'', q')$ for some $q'' \in \mathcal{Q}$ by Lemma 8.3.5. Thus, $(n' - n) \in a + b\mathbb{N}$ for some arithmetic progression $a + b\mathbb{N} \subseteq \Delta_{\uparrow}^{>0}(q'', q')$. Furthermore, let $c \in \mathbb{Z}/b\mathbb{Z}$ be the residue class of n' modulo b . So altogether, we will fix the witnesses q'' , $a + b\mathbb{N}$, and c in the following. First, let us first assume that $b > 0$. We now distinguish the cases when there are either (i) infinitely or (ii) finitely many $n'' \in \mathbb{N}$ such that $n'' \equiv c \pmod{b}$ and $(q', n'') \models \phi_{j'}$.

- Case (i) is expressed by the formula $\gamma(q'', a, b, c)_{\infty}$ which is defined as

$$\inf(q', j', c, b) \wedge \geq \nabla(q, q'') \wedge \equiv (c - a) \pmod{b}.$$

- Case (ii) can be realized by saying that the maximal such n'' is reachable from n via the arithmetic progression $a + b\mathbb{N}$. For this, let the formula $\gamma(q'', a, b, c)_{<\infty}$ be defined as the conjunction of

$$\neg \inf(q', j', c, b) \wedge \geq \nabla(q, q'')$$

and

$$\equiv (c - a) \pmod{b} \wedge \leq \max(\alpha_{(q', j')} - a, t_{(q', j')}).$$

The last conjunct guarantees that $n + a \leq n''$, which is necessary since we have to have that $n'' \in n + a + b\mathbb{N}$.

The case when $b = 0$ can easily be realized by putting

$$\gamma(q'', a) = \geq \nabla(q, q'') \wedge (\alpha_{(q', j')} - a).$$

Altogether, we put

$$\beta_1(q, q') = \bigvee_{q'' \in \mathcal{Q}} \left(\bigvee_{a + 0\mathbb{N} \subseteq \Delta_{\uparrow}^{>0}(q, q'')} \gamma(q'', a) \vee \bigvee_{\substack{a + b\mathbb{N} \subseteq \Delta_{\uparrow}^{>0}(q, q'') \\ b > 0, c \in \mathbb{Z}/b\mathbb{Z}}} \gamma(q'', a, b, c)_{\infty} \vee \gamma(q'', a, b, c)_{<\infty} \right).$$

Let us now consider case (2), i.e. when π is positive and $n \geq n'$. Then $n' \geq \nabla(q'', q')$ and $n - n' \in \Delta_{\downarrow}^{>0}(q, q'')$ for some $q'' \in Q$ by Lemma 8.3.5. Hence, $n - n' \in a + b\mathbb{N}$ for some $a + b\mathbb{N} \subseteq \Delta_{\downarrow}^{>0}(q, q'')$. Firstly, let us assume that $b > 0$. Recall that $c \in \mathbb{Z}/b\mathbb{Z}$ is the residue class of n' . Now the simple observation is that the witness n' can be replaced by the minimal $n'' \in \mathbb{N}$ such that $n'' \equiv n' \equiv c \pmod{b}$, $n'' \geq \nabla(q'', q')$, and $(q', n'') \models \phi_{j'}$. We realize this by the formula $\theta(q'', a, b, c)$ defined as the conjunction of $\equiv c + a \pmod{b}$ and

$$\geq \min \left((\geq \nabla(q'', q') \wedge \alpha_{(q', j')} \wedge \equiv c \pmod{b}) + a \right).$$

Secondly, let us assume that $b = 0$. This case is realized by the formula

$$\theta(q'', a) = \geq (\nabla(q'', q') + a) \wedge (\alpha_{(q', j')} + a).$$

Altogether, we define $\beta_2(q, q')$ to be

$$\bigvee_{q'' \in Q} \left(\bigvee_{\substack{a+b\mathbb{N} \subseteq \Delta_{\downarrow}^{>0}(q, q'') \\ b>0, c \in \mathbb{Z}/b\mathbb{Z}}} \theta(q'', a, b, c) \vee \bigvee_{a+0\mathbb{N} \subseteq \Delta_{\downarrow}^{>0}(q, q'')} \theta(q'', a) \right).$$

Finally, let us consider case (3), i.e. when π is zero. For each $q'' \in Q$, define the predicate

$$\exists m \in \mathbb{N} : m \in \Delta_{\uparrow}^{=0}(q'', q') \wedge (q', m) \models \phi_{j'}.$$

In other words, case (3) can be rephrased as $n \in \Delta_{\downarrow}^{=0}(q, q'')$ and $\pi(q'')$ for some $q'' \in Q$ by Lemma 8.3.4. Now check that the predicate $\pi(q'')$ can be expressed as

$$\begin{aligned} & \bigvee_{\substack{a+b\mathbb{N} \subseteq \Delta_{\uparrow}^{=0}(q'', q') \\ b>0}} 0 \leq \max(\equiv a \pmod{b} \wedge \alpha_{(q', j')}, t_{(q', j')} + L_{(q', j')}) \\ & \vee \bigvee_{a+0\mathbb{N} \subseteq \Delta_{\uparrow}^{=0}(q'', q')} a = \min((\alpha_{(q', j')} - a) + a). \end{aligned}$$

Define $\beta_3(q, q') = \bigvee_{q'' \in Q} \pi(q'') \wedge \rho(q'')$, where $\rho(q'')$ is

$$\bigvee_{a+b\mathbb{N} \subseteq \Delta_{\uparrow}^{=0}(q, q'')} (\equiv a \pmod{b}) \vee \bigvee_{a+0\mathbb{N} \subseteq \Delta_{\uparrow}^{=0}(q, q'')} (= a).$$

We finally put $\alpha_{(q, j)} = \bigvee_{q' \in Q} \beta_1(q, q') \vee \beta_2(q, q') \vee \beta_3(q, q')$.

This concludes the definition of α . It is straightforward to see that α can be computed in time polynomial in $\|\mathcal{P}\| + \|\phi\|$. By additionally applying Lemma 8.2.2, we obtain Theorem 8.4.1.

It turns out that a more precise analysis of our translation allows us to derive the following polynomial time upper bound, when the one-counter process is fixed.

Theorem 8.4.3 *The problem of model checking EF dag-formulas over a fixed OCP is in P.*

This is in stark contrast to the problem of model checking EF-logic over a fixed push-down system, which can be shown to be PSPACE-complete using the proof from [BEM97, Wal00]. The proof of Theorem 8.4.3 can be found in the appendix.

8.5 Application to weak-bisimilarity checking

In this section, we use our P^{NP} upper bound from the previous section to derive a P^{NP} upper bound for weak-bisimilarity checking of OCPs against finite systems. This improves the previous PSPACE upper bound known for the problem [Kuc00]. In the next section, we shall in fact give a P^{NP} lower bound for weak-bisimilarity checking of OCPs against finite systems.

We shall first recall the definition of weak bisimulation (cf. [KJ06]). Let $\mathfrak{S} = \langle S, \{\rightarrow_a\}_{a \in \text{ACT}} \rangle$ be a transition system and assume some distinguished internal symbol $\tau \in \text{ACT}$. Define the *extended transition relation* $\Rightarrow_a \subseteq S \times S$ for each $a \in \text{ACT}$ as $s \Rightarrow_a t$ if and only if either $a \neq \tau$ and there exist $s', t' \in S$ such that $s \xrightarrow{\tau}^* s' \xrightarrow{a} t' \xrightarrow{\tau}^* t$, or $a = \tau$ and $s \xrightarrow{\tau}^* t$. Given two transition systems $\mathfrak{S} = \langle S, \{\rightarrow_a\}_{a \in \text{ACT}} \rangle$ and $\mathfrak{S}' = \langle S', \{\rightarrow'_a\}_{a \in \text{ACT}} \rangle$, a relation $R \subseteq S \times S'$ is a *weak-bisimulation* (or *w-bisimulation*) if it is nonempty, and whenever $(s, s') \in R$, then for every $a \in \text{ACT}$ the following two conditions hold (i) if $s \xrightarrow{a} t$ for some $t \in S$, then $s' \Rightarrow'_a t'$ for some $t' \in S'$ such that $(t, t') \in R$, and (ii) if $s' \xrightarrow{a} t'$ for some $t' \in S'$, then $s \Rightarrow_a t$ for some $t \in S$ such that $(t, t') \in R$. For every $s \in S$ and every $s' \in S'$, we say that s and s' are *w-bisimilar*, written $s \approx s'$, whenever there exists a w-bisimulation $R \subseteq S \times S'$ such that $(s, s') \in R$. We now define *w-bisimilarity checking* of one-counter processes against finite systems as follows.

W-BISIMILARITY CHECKING: OCPs AGAINST FINITE SYSTEMS

Instance: A one-counter process \mathcal{P} , a state (q, n) of $\mathfrak{S}_{\mathcal{P}}$ with n given in binary, a finite system T , and a state t of T .

Question: $(q, n) \approx t$?

In order to derive our upper bound, we recall the following well-known result from [JKM01] (a more recent presentation can be found in [KJ06]).

Lemma 8.5.1 *Let $\mathfrak{S}_1 = \langle S_1, \{\xrightarrow{1}_a\}_{a \in \text{ACT}} \rangle$ be a (possibly infinite) transition system and $\mathfrak{S}_2 = \langle S_2, \{\xrightarrow{2}_a\}_{a \in \text{ACT}} \rangle$ be a finite transition system with k states. Then, given any*

state $s_2 \in S_2$, we can construct an EF dag-formula $\Phi_{s_2, \mathfrak{S}_2}$ in polynomial time (in k) such that, for every state $s_1 \in S_1$, it is the case that $s_1 \approx s_2$ if and only if $\mathfrak{S}_1, s_1 \models \Phi_{s_2, \mathfrak{S}_2}$.

In other words, Lemma 8.5.1 implies that the w-bisimulation checking problem is polynomial-time reducible to the problem of model checking EF dag-formulas. Combining this lemma with our results in the previous sections, the following theorem can easily be derived.

Theorem 8.5.2 *The w-bisimilarity checking problem is solvable in P^{NP} . The problem becomes solvable in P when the one-counter process is fixed.*

8.6 Lower bounds

We conclude this chapter with several tight lower bounds for the problem of model checking EF-logic over OCPs and weak-bisimulation problems of OCPs against finite systems.

Let us start with the problem of model checking EF-logic over OCPs. We have managed to give a P^{NP} lower bound for this problem in [GMT09] when the input formulas are represented as dags. This is achieved by a simple reduction from the P^{NP} -complete problem called DSAT. On the other hand, we will not reproduce the proof here since a matching P^{NP} lower bound has recently been given by Göller and Lohrey [GL10] for EF tree-formulas.

Proposition 8.6.1 ([GMT09, GL10]) *The problem of model checking EF dag/tree formulas over OCPs is P^{NP} -hard.*

We now proceed to the lower bound of weak-bisimilarity checking of OCPs against finite systems. We will show that this problem is P^{NP} -hard by a reduction from a problem called DSAT [Pap94], which takes the following input: a sequence of boolean formulas F_1, \dots, F_n with variables x_1, \dots, x_n and sets of variables Z_1, \dots, Z_n such that the formula F_i can take only variables from $\{x_1, \dots, x_{i-1}\}$ and Z_i . The goal is to decide whether there exists an assignment $\sigma : \{x_1, \dots, x_n\} \rightarrow \{0, 1\}$ that sets x_n to 1 such that the following are satisfied for all $i \in [n]$:

$$\sigma(x_i) = 1 \quad \Leftrightarrow \quad \exists Z_i F_i(x_1, \dots, x_{i-1}, Z_i). \quad (8.1)$$

Notice that imposing the constraint in (8.1) ensures that there exists a *unique* assignment σ . The only question is whether this assignment satisfies $\sigma(x_n) = 1$.

Proposition 8.6.2 *The problem of checking weak-bisimulation between a given one-counter net and a given finite system is hard for P^{NP} .*

To prove this proposition, we will have to be able to encode assignments to boolean formulas (i.e. a sequence of 0s and 1s) as numbers. We shall achieve this via *Gödel encodings* (i.e. using the uniqueness of prime decomposition). To this end, we shall define the Gödel function $\mathfrak{G} : \mathbb{Z}_{>0} \rightarrow \{0, 1\}^\omega$ as follows: if p_i is the i th prime number and $n = \prod_{i>0} p_i^{j_i}$, where $j_i \in \mathbb{N}$, then define $\mathfrak{G}(n) = j'_1 j'_2 \dots$, where $j'_i = 0$ if $j_i = 0$ and $j'_i = 1$ if $j_i > 0$. Since the proof of Proposition 8.6.2 is rather intricate, we shall only present a sketch of the proof (the full proof can be found in the appendix).

Proof Sketch. In this proof, we shall present the weak-bisimulation as games between two players: Attacker and Defender (e.g. see [Sti98]). Briefly, Attacker's goal is to prove that two given processes are not w-bisimilar, while Defender tries to prove otherwise. 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 \rightarrow_a . Defender must imitate this by moving the pebbled state from the other system to one of its successors by the same action \rightarrow_a , possibly together with several internal τ -actions, i.e. he has to move the pebble along a \Rightarrow_a -transition. If one player cannot move, then the other player wins. Defender wins every infinite game. Two states s and t are w-bisimilar (resp. not w-bisimilar) if and only if Defender (resp. Attacker) has a winning strategy on the game with initial pebble configuration (s, t) .

We now present the proof sketch of Proposition 8.6.2. We reduce DSAT to the w-bisimulation checking problem by constructing a suitable one-counter net and a finite system. The finite system contains (among many others) states P_1 and P_2 , and the one-counter net contains control-states s_{F_i} for $1 \leq i \leq n$ such that the following statements are equivalent:

- $\langle F_1, \dots, F_i \rangle \in \text{DSAT}$.
- $P_1 \approx s_{F_i}(l)$ for all $l \in \mathbb{N}$.
- P_2 is not w-bisimilar to $s_{F_i}(l)$ for some l in \mathbb{N} .

In the particular case of $i = n$, this is the reduction we are looking for. The idea is that checking the truth of every subformula $\exists Z_i F_i(x_1, \dots, x_{i-1}, Z_i)$ of the DSAT problem is encoded into a complex w-bisimulation game for $P_1 \approx s_{F_i}(l)$. In this game, the

defender player gets to choose (by a long $\xRightarrow{\tau}$ move) a natural number l' which is stored in the one-counter net. This number encodes (by Gödel encoding) the assignment of values to the boolean variables in the block Z_i . Later in the game, these values can be tested by special counter-decreasing loops, which implement divisibility tests on l' . The variables x_k are treated differently, because they depend on other subformulae F_k with smaller index numbers $k < i$. If the value of some x_k (for $k < i$) needs to be tested, then the w-bisimulation game jumps to some subgame which tests either $P_1 \approx s_{F_k}(l')$ or $P_2 \approx s_{F_k}(l')$, depending on whether the value of x_k is claimed as true or false.

The main technical difficulty of the proof is to restrict the freedom of the players in the w-bisimulation game, so that they exactly make the choices needed in the verification game for the formula $\exists Z_i F_i(x_1, \dots, x_{i-1}, Z_i)$ in the right step, and do not make a move that is reserved for the other player. This is rather intricate, because of the asymmetry of the two compared systems, one infinite-state one-counter process and a finite system. In particular, isomorphic copies of the finite system are replicated in the finite-control of the one-counter net. \square

In contrast to Kučera's DP lower bound [Kuc00] which holds for a fixed one-counter net, our proof of Proposition 8.6.2 requires that the finite system is not fixed. Nevertheless, we can show that the w-bisimilarity checking problem for fixed finite systems is harder than DP.

Proposition 8.6.3 *There exists a fixed finite system for which the w-bisimilarity checking problem is hard for $P^{NP[\log]}$ even for one-counter nets.*

This lower bound proof, which is given in the appendix, is achieved by a reduction from a $P^{NP[\log]}$ -complete problem called INDEX-ODD. Together with Lemma 8.5.1, we also obtain a $P^{NP[\log]}$ lower bound for a fixed EF formula (in this case tree/dag representations are not important since the formula is fixed).

Proposition 8.6.4 *There exists a fixed EF formula ϕ such that model checking ϕ over one-counter nets is $P^{NP[\log]}$ -hard.*

In [GMT09], we gave a direct reduction which shows the same hardness result when only EF_Γ operators satisfying $\Gamma = \text{ACT}$ are permitted.

Chapter 9

Networks of one-counter processes

In the previous chapter, we have studied one-counter processes and several verification problems over such systems. One-counter processes are, however, a rather weak model. They can only be used to model programs with bounded recursions and one unbounded integer variable (used as a counter). Let us now imagine a more general scenario when several integer-valued counters are used in a program. Consider the following code snippet written in a C-like language:

```
INT i, n, m;    // Initialized elsewhere
STRING s;      // bounded length
L1:
    n, s are used ...

    FOR( i = n; i ≥ 0; i = i − 1 )    // Synch.
        i, s are used ...

    m = n − 2;    // Synch.
L2:    ...
```

This program has three unbounded integer variables and other bounded data structures (e.g. strings). The places where these variables synchronize in the form of simple variable assignments are underlined. Notice that the synchronizations take place outside for/while loops. Within for/while loops, at most one integer variable is used. Observe that such a program already cannot be modeled by OCPs since we have to keep track of several integer counters simultaneously, e.g., the variable n in this program is assigned to i and m at different points of the program.

	FO(Reach) FO ⁴ (Reach)	FO ² (Reach)	EF-logic
Combined	PSPACE	PSPACE	in P ^{NP} & P ^{NP[log]} -hard (Chap. 8)
Expression	PSPACE	in P	in P (Chap. 8)
Data	PH	PH	in P ^{NP} & P ^{NP[log]} -hard (Chap. 8)

Table 9.1: Results for OCPs.

The approach that we will take in this chapter is to study model checking problems over *asynchronous products of OCPs*, which we denote by ΠOCP . They can also be construed as *networks of one-counter processes* with each component behaving independently (i.e. the processes do not interact). Observe that reachability for ΠOCP can trivially be reduced to the reachability problems of their components. On the other hand, when we consider the model checking problems of asynchronous products of OCPs with more powerful logics like FO(Reach) and EF-logic, asynchronous products are sufficiently powerful for modeling bounded synchronizations among the finite-control units of the OCPs. This follows from a more general result by Wöhrle and Thomas [WT07] regarding *finitely synchronized products* of infinite-state systems. In order to permit some synchronizations amongst the counters, we will also enrich these logics with some simple synchronization predicates. These will enable us to verify interesting properties of programs with multiple integer-valued counters with *bounded synchronizations* between the counters, i.e., in *every* execution of the program, the number of times at which integer variables synchronize is bounded a priori. Notice that the example of a program that we saw above falls within this category since it uses only two synchronizations outside for/while loops.

The model checking problems over ΠOCP that we will consider in this chapter are with respect to the specifications in (1) FO(Reach), (2) the k -variable fragments FO ^{k} (Reach) ($k \geq 2$) of FO(Reach), and (3) EF-logic, which is a fragment of FO²(Reach). We also study these logics extended with simple component-wise synchronizing predicates testing whether components i and j have the same counter values, which we denote by FO_S(Reach), FO_S ^{k} (Reach), and EF_S-logic, respectively. The goal of this chapter is to provide not only decidability results, but also the precise complexity of these model checking problems.

The results of this chapter are summarized in Table 9.1 and Table 9.2 together with the results from the previous chapter. Notice that OCPs are simply the special case of

	FO(Reach) FO ⁴ (Reach) FO _S (Reach)	FO ² (Reach)	EF-logic	EF _S -logic
Combined Expression Data	PSPACE PSPACE PH	PSPACE in P PH	PSPACE in P in P ^{NP} & P ^{NP[log]} -hard	PSPACE PSPACE PH

Table 9.2: Results for Π OCP.

Π OCP with only one component. In particular, all our results are within PSPACE, in contrast to PDS whose expression complexity for FO(Reach) is nonelementary [CH90]. Our upper bounds are shown by first introducing two syntactic restrictions \mathcal{L} and \mathcal{L}' of Presburger Arithmetic, for which we give optimal quantifier elimination procedures, and showing that the Π OCP model checking problems are poly-time reducible to either \mathcal{L} or \mathcal{L}' . Note that, to obtain a sharp upper bound, we cannot consider only OCPs (without products) and apply *Feferman-Vaught type of composition methods* (e.g. see [Mak04, Rab07, WT07]) as the resulting algorithm will run in time that is nonelementary in the formula size. Concerning our lower bound results, in contrast to the result from the previous chapter that model checking EF-logic is in P^{NP}, data complexity of FO²(Reach) over OCPs is already hard for every level of PH. On the other hand, the expression complexity of FO²(Reach) over Π OCP is in P. This generalizes one of the key results from the previous chapter that the expression complexity of EF-logic over OCPs (without products) is in P. However, for each $k > 3$, we can show that the expression complexity of FO^k(Reach) is PSPACE-complete already for OCPs. Also, notice that the combined complexity of EF-logic becomes PSPACE, which holds already for products of two OCPs. Finally, notice that adding simple synchronization relations to EF-logic causes the expression and data complexity to increase significantly.

What about model checking with respect to specifications in CTL or LTL? We do not consider them since they are easily shown to be undecidable by a simple reduction from reachability of Minsky's 2-counter systems, e.g., we can encode full synchronizations of the counters in a CTL formula of the form $E(\phi \cup \psi)$ (synchronization is embedded in ϕ). For monadic second-order logic, undecidability can be more easily obtained by an asynchronous product of two OCPs which only increments their counters (i.e. this generates the two-dimensional infinite grid with an undecidable MSO

theory [Tho96]).

This chapter is organized as follows. We start with the definitions of asynchronous products of OCPs and logics with synchronizing predicates in Section 9.1. In Section 9.2, we define our two fragments \mathcal{L} and \mathcal{L}' of Presburger arithmetic and give a reduction from our model checking problems to the membership problems of these logics. In Section 9.3, we give optimal quantifier elimination procedures for \mathcal{L} and \mathcal{L}' and deduce optimal upper bounds for all model checking problems in Table 9.1 and Table 9.2. We prove the lower bounds in Section 9.4. The results in this chapter have previously appeared in [To09a].

9.1 Preliminaries

In this section, we first define the notions of asynchronous products of OCPs. We then explain how synchronizing predicates can be added in our logics.

9.1.1 Asynchronous products

We now review the definition of asynchronous products of transition systems and asynchronous products of OCPs.

Let $\text{ACT}_1, \dots, \text{ACT}_r$ be r pairwise disjoint sets of actions. Let ACT be their union. For each $i \in [1, r]$, let $\mathfrak{G}_i = \langle V_i, \{E_a\}_{a \in \text{ACT}_i} \rangle$ be a transition system over ACT_i . An *asynchronous product* of $\mathfrak{G}_1, \dots, \mathfrak{G}_r$ is the transition system $\Pi_{i=1}^r \mathfrak{G}_i := \langle V, \{\bar{E}_a\}_{a \in \text{ACT}} \rangle$, where $V := \Pi_{i=1}^r V_i$ and, whenever $a \in \text{ACT}_i$, $\bar{u} = (u_1, \dots, u_r)$, and $\bar{v} = (v_1, \dots, v_r)$, we have $(\bar{u}, \bar{v}) \in \bar{E}_a$ iff $(u_i, v_i) \in E_a$ and $u_j = v_j$ for all $j \neq i$. Intuitively, the product is “asynchronous” as each edge relation in $\Pi_{i=1}^r \mathfrak{G}_i$ changes at most one component in each vertex of $\Pi_{i=1}^r \mathfrak{G}_i$, i.e., causing no interaction between different components. See [Rab07, WT07] for more details.

An *asynchronous product* \mathcal{P} of r OCPs is simply a tuple of r OCPs $\mathcal{P}_1, \dots, \mathcal{P}_r$ over pairwise disjoint action alphabets $\text{ACT}_1, \dots, \text{ACT}_r$. The product \mathcal{P} has action labels $\text{ACT} := \text{ACT}_1 \cup \dots \cup \text{ACT}_r$. Then, the transition system $\mathfrak{S}_{\mathcal{P}}$ generated by \mathcal{P} is defined to be the transition system $\Pi_{i=1}^r \mathfrak{S}_{\mathcal{P}_i}$ over ACT . In the sequel, asynchronous products of OCPs are abbreviated as ΠOCP .

We define the model checking problems of ΠOCP with respect to $\text{FO}(\text{Reach})$ as follows.

MODEL CHECKING FO(Reach) OVER ΠOCP

Instance: A ΠOCP \mathcal{P} over ACT, an FO(Reach) formula $\varphi(x_1, \dots, x_k)$ over ACT, k configurations $\{\mathbf{c}_i\}_{i=1}^k$ of \mathcal{P} with binary representation of numbers.

Question: $\mathfrak{S}_{\mathcal{P}} \models \varphi(\mathbf{c}_1, \dots, \mathbf{c}_k)$?

For other logics like $\text{FO}^k(\text{Reach})$ ($k \geq 2$) and EF-logic, we could define the model checking problems in a similar fashion.

9.1.2 Synchronization predicates

So far, our logics cannot compare the values between two different counters in the system. We now introduce an extension that permits simple comparison tests (or *synchronizing predicates*) between counter values.

To add synchronizing predicates to the logic, we first add these predicates in the semantics of ΠOCP. Given an asynchronous product ΠOCP of r OCPs $\mathcal{P}_1, \dots, \mathcal{P}_r$ over $\text{ACT} = \bigcup_{i=1}^r \text{ACT}_i$, we define a transition system $\mathfrak{S}_{\mathcal{P}}^S$ as the transition system $\mathfrak{S}_{\mathcal{P}}$ expanded with the “synchronizing” edge relations $\{=_{i,j}\}_{1 \leq i \neq j \leq r}$ that are defined as

$$=_{i,j} := \{(\mathbf{c}, \mathbf{c}) : n_i = n_j\},$$

where $\mathbf{c} = ((q_1, n_1), \dots, (q_r, n_r))$. In other words, the relation $=_{i,j}$ contains all self-loops in $\mathfrak{S}_{\mathcal{P}}^S$ restricted to tuples in which the counter values of the i th component and the j th component agree. The graph $\mathfrak{S}_{\mathcal{P}}^S$ has action labels $\text{ACT} \cup \{(i, j)\}_{i,j \in [1,r]}$.

The logics $\text{FO}_S(\text{Reach})$, $\text{FO}_S^k(\text{Reach})$, and EF_S -logic are simply defined to be $\text{FO}(\text{Reach})$, $\text{FO}^k(\text{Reach})$, and EF -logic interpreted over this modified semantics. The problem of model checking $\text{FO}_S(\text{Reach})$ over ΠOCP can be defined as follows.

MODEL CHECKING $\text{FO}_S(\text{Reach})$ OVER ΠOCP

Instance: An asynchronous product \mathcal{P} of r OCPs ACT, an $\text{FO}_S(\text{Reach})$ formula $\varphi(x_1, \dots, x_k)$ over $\text{ACT} \cup \{(i, j)\}_{i,j \in [1,r]}$, k configurations $\{\mathbf{c}_i\}_{i=1}^k$ of \mathcal{P} with binary representation of numbers.

Question: $\mathfrak{S}_{\mathcal{P}}^S \models \varphi(\mathbf{c}_1, \dots, \mathbf{c}_k)$?

We can similarly define the model checking problems of $\text{FO}_S^k(\text{Reach})$ and EF_S -logic over ΠOCPs.

9.2 Two fragments \mathcal{L} and \mathcal{L}' of Presburger Arithmetic

We define our first fragment \mathcal{L} of Presburger Arithmetic, to which we will reduce the model checking of $\text{FO}_{\mathcal{S}}(\text{Reach})$ over ΠOCP .

Definition 9.2.1 *The syntax of the logic \mathcal{L} is as follows. Atomic propositions are of the form:*

- $x \sim y + c$, where $\sim \in \{\leq, \geq, =\}$,
- $x \sim c$, where $\sim \in \{\leq, \geq, =\}$,
- $x \equiv y + c \pmod{d}$, where $c \in [0, d - 1]$, and
- $x \equiv c \pmod{d}$, where $c \in [0, d - 1]$.

Here, x and y can take any variables, while c and d are constant natural numbers, given in binary representations. We then close the logic under boolean combinations, and existential and universal quantifications. The semantics is given directly from Presburger Arithmetic. The expression $x \equiv y + c \pmod{d}$ is to be interpreted as the Presburger formula $\exists z (x = y + c + dz \vee x + dz = y + c)$.

Intuitively, the logic \mathcal{L} is the fragment of Presburger Arithmetic that permits only inequality tests, addition with constants, and modulo tests. We now impose some further syntactic restrictions to our logic \mathcal{L} , to which model checking $\text{FO}^2(\text{Reach})$ over ΠOCP is still poly-time reducible.

Definition 9.2.2 *Define the logic \mathcal{L}' as follows. The only variables allowed are x_i and y_i , where $i \in \mathbb{Z}_{>0}$. The atomic propositions of \mathcal{L}' are given as follows for each $i \in \mathbb{Z}_{>0}$:*

- $x_i \sim y_i + c$ and $y_i \sim x_i + c$,
- $x_i \sim c$ and $y_i \sim c$,
- $x_i \equiv y_i + c \pmod{d}$ and $y_i \equiv x_i + c \pmod{d}$, and
- $x_i \equiv c \pmod{d}$ and $y_i \equiv c \pmod{d}$.

Here, c and d are constant natural numbers given in binary. We then close the logic under boolean combinations, and existential and universal quantifications.

Observe that the logic \mathcal{L}' allows only two variables x_i and y_i to be related. In fact, if we only allow x_1 and y_1 as variables, then \mathcal{L}' coincides with FO^2 fragment of \mathcal{L} .

We shall briefly discuss the expressive power of \mathcal{L} in terms of subsets of \mathbb{N}^k that can be defined in the logics. Let us first briefly recall the definition of *first-order modulo counting logic* FO_{MOD} , which extends FO with the modulo counting quantifiers $\exists^{p,q}$, for each $q \in \mathbb{Z}_{>0}$ and $p \in [0, q)$. When interpreted over $(\mathbb{N}, <)$, the semantics of FO_{MOD} is defined over $(\mathbb{N}, <)$ as follows: $(\mathbb{N}, <) \models \exists^{p,q} x \varphi(x, \bar{b})$ iff the number $l := |\{a \in \mathbb{N} : (\mathbb{N}, <) \models \varphi(a, \bar{b})\}|$ is either infinite or finite and $l \equiv p \pmod{q}$. See [Pél92] for more details. It turns out that \mathcal{L} coincides with the FO_{MOD} theory over $\langle \mathbb{N}, < \rangle$. In fact, [Pél92] shows that FO_{MOD} theory over $\langle \mathbb{N}, < \rangle$ admits a quantifier elimination, when the vocabulary is expanded with congruence tests. Therefore, \mathcal{L} subsumes FO_{MOD} over $\langle \mathbb{N}, < \rangle$. To show that $\mathcal{L} \subseteq \text{FO}_{\text{MOD}} \langle \mathbb{N}, < \rangle$, observe that expressions of the form $x \sim y + c$ can easily be replaced by equivalent FO formulas over $\langle \mathbb{N}, < \rangle$. Also, the atomic formula $x \equiv y + c \pmod{d}$ can be defined as $\bigwedge_{a=0}^{d-1} (y \equiv a \pmod{d} \leftrightarrow x \equiv a + c \pmod{d})$, and congruence tests $x \equiv a \pmod{d}$ can be defined in FO_{MOD} over $\langle \mathbb{N}, < \rangle$ as $\exists^{a,d} y (y < x)$. The expressive power of FO_{MOD} over $\langle \mathbb{N}, < \rangle$ was shown in [Pél92] to be strictly in between FO over $\langle \mathbb{N}, < \rangle$ and Presburger Arithmetic. For example, it was shown that Presburger formulas of the form $x = 2y$ is not definable in FO_{MOD} over $\langle \mathbb{N}, < \rangle$. Finally, we shall emphasize that the proof in [Pél92] of quantifier elimination for FO_{MOD} over $\langle \mathbb{N}, < \rangle$ expanded with congruence tests is nonconstructive.

The *membership problem of the logic \mathcal{L}* is defined as follows: given $\varphi(\bar{x}) \in \mathcal{L}$, where $\bar{x} = (x_1, \dots, x_n)$ and a tuple $\bar{a} \in \mathbb{N}^n$ in binary, decide whether $\langle \mathbb{N}, + \rangle \models \varphi(\bar{a})$. The membership problem for \mathcal{L}' can be defined similarly. We now state a proposition, which gives a reduction from model checking problems of ΠOCP to the membership problem for \mathcal{L} or \mathcal{L}' .

Proposition 9.2.1 *There is a poly-time reduction from the problem of model checking $\text{FO}_{\mathcal{S}}(\text{Reach})$ (resp. $\text{FO}^2(\text{Reach})$) over ΠOCP to the membership problem for \mathcal{L} (resp. \mathcal{L}'). Furthermore, the alternation rank of the output formula in \mathcal{L} (resp. \mathcal{L}') is the same as the alternation rank of the input formula in $\text{FO}_{\mathcal{S}}(\text{Reach})$ (resp. $\text{FO}^2(\text{Reach})$) up to addition by a small constant.*

This proposition can be proved easily using Lemma 8.3.1, Lemma 8.3.4, and Lemma 8.3.5. For this reason, we relegate the proof to the appendix.

9.3 Complexity upper bounds

In this section, we shall show that the combined and data complexity of $\text{FO}_S(\text{Reach})$ over ΠOCP are, respectively, in PSPACE and PH. We then show that the expression complexity of $\text{FO}^2(\text{Reach})$ is in P. To deduce a P^{NP} upper bound for data complexity of EF-logic over ΠOCP , it suffices to invoke the Feferman-Vaught type of composition method for EF-logic [Rab07] and use the P^{NP} algorithm for model checking EF-logic over OCPs from the previous chapter. Observe that these will give the claimed upper bounds in Table 9.1 and Table 9.2.

9.3.1 Combined and data complexity of $\text{FO}_S(\text{Reach})$

We start with the combined and data complexity of $\text{FO}_S(\text{Reach})$ over ΠOCP .

Theorem 9.3.1 *The combined and data complexity $\text{FO}_S(\text{Reach})$ over ΠOCP are in PSPACE and in PH, respectively.*

By Proposition 9.2.1, to deduce this theorem it suffices to prove the following proposition.

Proposition 9.3.2 *The membership problem of \mathcal{L} -formulas is in PSPACE. Moreover, fixing the alternation rank of input formulas, the problem is in PH.*

The proof is done via a quantifier elimination technique (e.g. see [Koz06] for an overview). Intuitively, our proof can be thought of as an extension of Ehrenfeucht-Fraïssé games on linear orders (e.g. see [Lib04]) with modulo tests. We first define an equivalence relation $\equiv_{p,m}^k$ on tuples of natural numbers.

Definition 9.3.1 *Given two $(k+1)$ -tuples $\bar{a} = (a_0, \dots, a_k), \bar{b} = (b_0, \dots, b_k)$ of natural numbers such that $a_0 = b_0 = 0$ and two numbers $p, m > 0$, we write $\bar{a} \equiv_{p,m}^k \bar{b}$ iff for all $i, j \in [0, k]$ the following statements hold:*

1. $|a_i - a_j| < pm$ implies $|a_i - a_j| = |b_i - b_j|$,
2. $|b_i - b_j| < pm$ implies $|a_i - a_j| = |b_i - b_j|$,
3. $|a_i - a_j| \geq pm$ iff $|b_i - b_j| \geq pm$,
4. $a_i \equiv b_i \pmod{p}$,
5. $a_i \leq a_j$ iff $b_i \leq b_j$.

The first two conditions above state that if two elements are “near”, then the difference for the corresponding two elements in the other tuple is the same. The third condition is the opposite of this condition: if two elements are “far” away from each other, then so are the corresponding two elements in the other tuple. It is easy to check that, given $m' \geq m > 0$, we have $\bar{a} \equiv_{p,m'}^k \bar{b}$ implies $\bar{a} \equiv_{p,m}^k \bar{b}$. Similarly, if $p|p'$, then $\bar{a} \equiv_{p',m}^k \bar{b}$ implies $\bar{a} \equiv_{p,m}^k \bar{b}$. The following lemma can be used to eliminate a quantifier.

Lemma 9.3.3 *Given two $(k+1)$ -tuples $\bar{a} = (a_0, \dots, a_k), \bar{b} = (b_0, \dots, b_k)$ of natural numbers such that $a_0 = b_0 = 0$ and two numbers $p, m > 0$, if $\bar{a} \equiv_{p,3m}^k \bar{b}$, then for all $a' \in \mathbb{N}$, there exists $b' \in \mathbb{N}$ such that $\bar{a}, a' \equiv_{p,m}^{k+1} \bar{b}, b'$.*

Proof. In this proof, for two numbers $c, d \in \mathbb{N}$, we write $d(c, d)$ to denote $|c - d|$. Suppose that $a' \in \mathbb{N}$. If $a' = a_i$ for some $i \in [0, k]$, then we simply set $b' = b_i$ and see that $\bar{a}, a' \equiv_{p,m}^{k+1} \bar{b}, b'$. Otherwise, there are two cases to consider:

(Case I) the number a' falls into a region $R = (a_r, a_s)$ for some distinct integers $r, s \in [0, k]$.

(Case II) the number a' falls into a region $R = (a_r, \infty)$ for some $r \in [0, k]$ such that there is no $s \in [0, k]$ with $a_r < a_s$.

Let us first consider Case I. Pick two indices r and s such that there is no $l \in [0, k]$ with $a_l \in R$. There are several subcases to consider:

1. $d(a_r, a_s) < 3pm$. In this case, assumption implies that $d(b_r, b_s) = d(a_r, a_s)$. Therefore, we may pick $b' = b_r + d(a_r, a')$. It is then easy to verify that $\bar{a}, a' \equiv_{p,m}^{k+1} \bar{b}, b'$.

2. $d(a_r, a_s) \geq 3pm$. In this case, our assumption implies that $d(b_r, b_s) \geq 3pm$. There are now three further subcases to consider:

(a) $d(a_r, a') < pm$. In this case, it follows that we have $d(a', a_s) \geq pm$. Pick $b' = b_r + d(a_r, a')$. It is then easy to check that $\bar{a}, a' \equiv_{p,m}^{k+1} \bar{b}, b'$.

(b) $d(a', a_s) < pm$. This case is similar to the previous item; one can set $b' := b_s - d(a', a_s)$.

(c) $d(a_r, a') \geq pm$ and $d(a', a_s) \geq pm$. Since $d(a_r, a_s) \geq 3pm$, it follows that there are at least p consecutive numbers in the region $[a_r + pm, a_s - pm]$. Likewise, since $d(b_r, b_s) \geq pm$, there are at least p consecutive numbers in

the region $[b_r + pm, b_s - pm]$. This implies that for every $c \in [0, p)$, there exists $a'' \in (a_r, a_s)$ with $a'' \equiv c \pmod{p}$, $d(a_r, a'') \geq pm$, and $d(a'', a_s) \geq pm$ iff there exists $b'' \in (b_r, b_s)$ with $b'' \equiv c \pmod{p}$, $d(b_r, b'') \geq pm$ and $d(b'', b_s) \geq pm$. Therefore, if $a' \equiv c \pmod{p}$ for some $c \in [0, p)$, it follows that there exists $b' \in (b_r, b_s)$ with $b' \equiv c \pmod{p}$, $d(b_r, b') \geq pm$ and $d(b', b_s) \geq pm$. It is easy now to check that $\bar{a}, a' \equiv_{p,m}^{k+1} \bar{b}, b'$.

This completes our proof for Case I. Let us now turn to Case II. There are two possibilities:

1. $d(a_r, a') < pm$. In this case, we set $b' = b_r + d(a_r, a')$ and it is easy to see that $\bar{a}, a' \equiv_{p,m}^{k+1} \bar{b}, b'$.
2. $d(a_r, a') \geq pm$. In this case, we may set $b' = b_r + pm + (d(a_r, a') \bmod p)$. Since we have $a_r \equiv b_r \pmod{p}$, it follows that $a' \equiv b' \pmod{p}$. It is easy now to verify that $\bar{a}, a' \equiv_{p,m}^{k+1} \bar{b}, b'$.

This concludes our proof. \square

Let us consider only tuples $\bar{a} = (a_0, \dots, a_k)$ of natural numbers satisfying $a_0 = 0$. Given an $\equiv_{p,3m}^k$ -equivalence class C and an $\equiv_{p,m}^{k+1}$ -equivalence class C' , we say that C' is *consistent with* C if there exist a tuple $\bar{a} = (a_0, \dots, a_k)$ of natural numbers and a number $a' \in \mathbb{N}$ such that $a_0 = 0$, $\bar{a} \in C$, and $(\bar{a}, a') \in C'$. The following lemma shows that we need not consider large numbers when eliminating a quantifier.

Lemma 9.3.4 *Let $\bar{a} = (a_0, \dots, a_k)$ be a tuple of natural numbers and C be its $\equiv_{p,3m}^k$ -equivalence class. Then, every $\equiv_{p,m}^{k+1}$ -equivalence class has a representative in the set $\{(\bar{a}, a') : 0 \leq a' \leq \max(\bar{a}) + pm + p\}$.*

Proof. This follows from the proof of Lemma 9.3.3, i.e., that we never need to add more than $pm + p$ from the maximal element in \bar{b} . \square

Define $r(0, m) := m$ and $r(n+1, m) := 3r(n, m)$, for $n \in \mathbb{N}$. By induction, we have $r(n, m) = 3^n m$. Let us now define the notion of *offsets* and *periods* of formulas in \mathfrak{L} . If ϕ are atomic formulas of the form $x \sim y + c$, $x \sim c$, $x \equiv y + c \pmod{d}$, or $x \equiv c \pmod{d}$, then *offsets* of ϕ are defined to be the integer c . If ϕ is not an atomic formula, then its *offset* is the largest offset of atomic subformulas of ϕ . If ϕ are atomic formulas of the form $x \sim y + c$ or $x \sim c$, then its *period* is defined to be 1. If ϕ are atomic formulas of the form $x \equiv y + c \pmod{d}$ or $x \equiv c \pmod{d}$, then its *period* is defined

to be d . Otherwise, if φ is not an atomic formula, its *period* is defined to be the least common multiple of the periods of each of its atomic subformulas. For $p, m \in \mathbb{Z}_{>0}$, define $\mathcal{L}_{p,m}$ to be formulas in \mathcal{L} , whose periods divide p and whose offsets are smaller than m .

Lemma 9.3.5 *Let $p, m \in \mathbb{Z}_{>0}$. Suppose $\bar{a} = (a_0, \dots, a_k), \bar{b} = (b_0, \dots, b_k)$ are tuples of natural numbers satisfying $a_0 = b_0 = 0$ and $\bar{a} \equiv_{p,r(n,m)}^k \bar{b}$. Then, given a formula $\varphi(x_1, \dots, x_k)$ in $\mathcal{L}_{p,m}$ of quantifier rank n ,*

$$\langle \mathbb{N}, + \rangle \models \varphi(a_1, \dots, a_k) \Leftrightarrow \langle \mathbb{N}, + \rangle \models \varphi(b_1, \dots, b_k).$$

Proof. The proof is by induction on φ . Let us consider the base cases. There are four cases:

- φ is of the form $x_i \sim x_j + c$. Suppose that $d(a_i, a_j) < pm$. In this case, our assumption $\bar{a} \equiv_{p,m}^k \bar{b}$ implies that we have $d(a_i, a_j) = d(b_i, b_j)$. Pick an integer r such that $a_i = b_i + r$. Then, since $a_i \leq a_j \Leftrightarrow b_i \leq b_j$, we have $a_j = b_j + r$. Therefore, for each $\sim \in \{\leq, \geq, =\}$, we have $a_i \sim a_j + c$ iff $b_i + r \sim b_j + r + c$ iff $b_i \sim b_j + c$. Let us now consider the case when $d(a_i, a_j) \geq pm$. In this case, our assumption $\bar{a} \equiv_{p,m}^k \bar{b}$ implies that $d(b_i, b_j) \geq pm$. Then, since $a_i \leq a_j \Leftrightarrow b_i \leq b_j$ and $c < m \leq pm$, it follows that $a_i \leq a_j + c \Leftrightarrow b_i \leq b_j + c$. Furthermore, since $c < m \leq pm$, it follows that $a_i \neq a_j + c$ and $b_i \neq b_j + c$. Altogether, these imply that $a_i \sim a_j + c$ iff $b_i \sim b_j + c$ for each $\sim \in \{\leq, \geq, =\}$.
- φ is of the form $x_i \sim c$. This follows from the proof for the previous case.
- φ is of the form $x_i \equiv x_j + c \pmod{d}$. Since $a_i \equiv b_i \pmod{p}$ and $a_j \equiv b_j \pmod{p}$, we also have $a_i \equiv b_i \pmod{d}$ and $a_j \equiv b_j \pmod{d}$ as d divides p . It is then immediate that $a_i \equiv a_j + c \pmod{d}$ iff $b_i \equiv b_j + c \pmod{d}$.
- φ is of the form $x_i \equiv c \pmod{d}$. Same as the previous case.

We now turn to the inductive cases. The cases for boolean combinations are easy. So, consider the case when φ is of the form $\exists x_{k+1} \psi(\bar{x}, x_{k+1})$. Then, let us prove that $\langle \mathbb{N}, + \rangle \models \varphi(\bar{a})$ implies $\langle \mathbb{N}, + \rangle \models \varphi(\bar{b})$; the converse is completely symmetric. If $\langle \mathbb{N}, + \rangle \models \varphi(\bar{a})$, then there exists $a_{k+1} \in \mathbb{N}$ such that $\langle \mathbb{N}, + \rangle \models \psi(\bar{a}, a_{k+1})$. Since $\bar{a} \equiv_{p,m}^k \bar{b}$, Lemma 9.3.3 implies that there exists $b_{k+1} \in \mathbb{N}$ such that $\bar{a}, a' \equiv_{p,r(n-1,m)}^{k+1} \bar{b}, b'$. By induction, it follows that $\langle \mathbb{N}, + \rangle \models \psi(\bar{b}, b')$, which proves that $\langle \mathbb{N}, + \rangle \models \varphi(\bar{b})$. \square

We are now ready to prove Proposition 9.3.2.

Proof of Proposition 9.3.2. We now give a polynomial-time alternating Turing machine M which checks whether $\langle \mathbb{N}, + \rangle \models \phi(a_1, \dots, a_n)$ for given a formula $\phi(x_1, \dots, x_n)$ and a $n+1$ -tuple $\bar{a} = (a_0, \dots, a_n)$, where $a_0 = 0$. First, push all the negations downward to the atomic propositions level, which can be done easily. Suppose that p and m be, respectively, the period and offset of the input formula. Now if ϕ is an atomic proposition (i.e. inequality, or modulo tests), it is easy to see that M can check it in poly-time. If ϕ is $\psi \vee \psi'$ (resp. $\psi \wedge \psi'$), then existentially (resp. universally) guess ψ or ψ' and check the guessed formula. If ϕ is of the form $\exists x \psi(\bar{y}, x)$ (resp. $\forall x \psi(\bar{y}, x)$) and has quantifier rank k , then M existentially (resp. universally) guesses a number a_{n+1} not exceeding $\max(\bar{a}) + pr(k, m) + p \leq \max(\bar{a}) + p3^k m + p$ and check whether $\langle \mathbb{N}, + \rangle \models \psi(\bar{a}, a_{n+1})$. The upper bound for a_{n+1} is sufficient due to Lemma 9.3.4.

To analyze the running time of M , notice that the maximum number that M can guess on any of its run on input ϕ of quantifier rank h and a tuple \bar{a} of natural numbers (in binary) is $\max(\bar{a}) + \sum_{j=0}^h (pr(j, m) + p) \leq \max(\bar{a}) + p(h+1)3^h m + p(h+1)$, which can be represented using polynomially many bits. [Note that p and m are represented in binary and so the guessed number is polynomial in $\log(p)$ and $\log(m)$.] This implies that membership of \mathcal{L} -formulas is in PSPACE. Finally, notice that the number of alternations used by M corresponds to the alternation rank of ϕ . Therefore, considering only formulas of fixed alternation rank, the membership problem for \mathcal{L} -formulas is in PH. \square

9.3.2 Expression complexity of $\text{FO}^2(\text{Reach})$

We now deal with the expression complexity of $\text{FO}^2(\text{Reach})$.

Theorem 9.3.6 *The expression complexity of $\text{FO}^2(\text{Reach})$ over ΠOCP is in P.*

Define $\mathcal{L}'_{p,m}$ to be the set of all formulas in \mathcal{L}' whose periods divide p and whose offsets do not exceed m . Let $\mathcal{L}'_{p,m}(n)$ to be the set of all formulas in $\mathcal{L}'_{p,m}$ that use only variables in $\{x_1, \dots, x_n\} \cup \{y_1, \dots, y_n\}$. For all fixed $p, m, n \in \mathbb{Z}_{>0}$, the *membership problem of $\mathcal{L}'_{p,m}(n)$* is as follows: given $\phi(\bar{x}, \bar{y}) \in \mathcal{L}'_{p,m}(n)$ and two tuples $\bar{a}, \bar{b} \in \mathbb{N}^n$ of numbers in binary representation, decide whether $\langle \mathbb{N}, + \rangle \models \phi(\bar{a}, \bar{b})$. By Proposition 9.2.1, Theorem 9.3.6 follows from the following proposition.

Proposition 9.3.7 *For fixed $p, m, n \in \mathbb{Z}_{>0}$, the membership problem of $\mathcal{L}'_{p,m}(n)$ is in P.*

This proposition can also be proved via quantifier elimination. The intuition that we can obtain a poly-time algorithm is from a two-pebble Ehrenfeucht-Fraïssé games over linear orders (see [Lib04]), which can only distinguish small linear orders (i.e. only linear in the quantifier rank of the $\text{FO}^2(\text{Reach})$ formula). The proof is similar to the case for $\text{FO}(\text{Reach})$, but is much more tedious, which we relegate to the appendix.

9.4 Complexity lower bounds

In order to facilitate our lower bound proofs in this section, we shall define a 2-player game, called *the buffer game*, which we shall prove to be PSPACE-complete. First, let \mathcal{L}_{DIV} be the set of quantifier-free \mathcal{L} -formulas in 3-CNF (i.e. in CNF and each clause has exactly three literals) with one free variable x , whose atomic propositions are of the form $x \equiv 0 \pmod{p}$ where p is a prime number. The buffer game is played by Player \exists and Player \forall . An arena of the buffer game is a tuple (\bar{v}, k, ϕ) , where \bar{v} is a finite and strictly increasing sequence of positive integers, k is the number of integers in \bar{v} , and ϕ a formula of \mathcal{L}_{DIV} . The buffer game with arena (\bar{v}, k, ϕ) , where $\bar{v} = (v_1, \dots, v_k)$, has $k + 1$ rounds and is played as follows. Each round r defines a *positive* number m_r , which represents the current buffer value. At round 0, Player \exists chooses a number m_0 to be written to the buffer. Suppose that $0 < r \leq k$, and m_0, \dots, m_{r-1} are the buffer values chosen from the previous rounds. At even (resp. odd) round r , Player \exists (resp. Player \forall) rewrites the buffer by a number $m_r \geq m_{r-1}$ of his choosing such that $m_r \equiv m_{r-1} \pmod{\prod_{j=1}^{v_r} p_j}$, i.e., $m_r = m_{r-1} + c \left(\prod_{j=1}^{v_r} p_j \right)$ for some $c \in \mathbb{N}$. In particular, by Chinese remainder theorem, this condition implies that, for each $1 \leq j \leq v_r$, $p_j | m_r$ iff $p_j | m_{r-1}$. In other words, each player is not allowed to “overwrite” some divisibility information in the buffer. Player \exists *wins* if $\langle \mathbb{N}, + \rangle \models \phi(m_k)$. Otherwise, Player \forall *wins*. The problem **BUFFER** is defined as follows: given an arena (\bar{v}, k, ϕ) of the buffer game, where *each number is represented in unary*, decide whether Player \exists has a winning strategy. For each $n \in \mathbb{N}$, we define the problem **BUFFER_n** to be the restriction of the problem **BUFFER** which takes only an input arena of the form (\bar{v}, n, ϕ) .

Lemma 9.4.1 *The problem **BUFFER** is PSPACE-complete. The problem **BUFFER_k** is Σ_{k+1}^P -complete.*

Loosely speaking, by applying Gödel encoding (see Section 8.6 for its definition) one can encode each truth valuation for boolean formulas into a number. Therefore, boolean formulas can be reduced to statements about divisibility. Furthermore, a *block*

of \exists (resp. \forall) quantifiers in a quantified boolean formula can be reduced into a choice of number at a single round in the buffer game for Player \exists (resp. Player \forall). The proof of Lemma 9.4.1 can be found in the appendix.

We now use the buffer game to prove our first lower bound result for the problem of model checking OCPs.

Proposition 9.4.2 *Combined complexity of $\text{FO}^2(\text{Reach})$ on OCPs is PSPACE-hard. For every $k \in \mathbb{N}$, there is a fixed formula ϕ_k of $\text{FO}^2(\text{Reach})$ with $k + c$ quantifier alternations, for some small constant $c \in \mathbb{N}$, such that checking ϕ_k over OCPs is Σ_k^P -hard.*

To prove this theorem, we first state a standard lemma, whose proof can be found in [JKMS04] (similar proof techniques have been used earlier in [Kuc00], and were also used in our lower bound proofs in the previous chapter).

Lemma 9.4.3 *Given a \mathcal{L}_{DIV} -formula ϕ , we can compute in polynomial time an OCP \mathcal{P} with a fixed set Γ of action symbols and an initial state q_I such that, for each positive integer m , it is the case that $\mathfrak{S}_{\mathcal{P}}, (q_I, m) \models \alpha$ iff $\langle \mathbb{N}, + \rangle \models \phi(m)$, where α is a small fixed EF formula.*

The crucial idea in the proof of the above lemma is that both divisibility and indivisibility tests of the form $p|x$ or $p \nmid x$ can be reduced to a certain reachability question for an appropriate OCP \mathcal{P} by embedding a cycle of length p in \mathcal{P} .

Proof sketch of Proposition 9.4.2. We give a poly-time reduction from BUFFER. Given an arena $\mathcal{A} = (\bar{v}, k, \phi)$, we compute an $\text{FO}^2(\text{Reach})$ sentence ϕ' , and a OCP $\mathcal{P} = (Q, \delta_0, \delta_{>0})$ such that Player \exists has a winning strategy in \mathcal{A} iff $\mathfrak{S}_{\mathcal{P}} \models \phi'$. Let $\bar{v} = (v_1, \dots, v_k)$. As we shall see, ϕ' depends only on k and has quantifier rank $k + c$ for some small constant $c \in \mathbb{N}$, which by Lemma 9.4.1 will prove the desired lower bound for data complexity.

We now run the algorithm given by Lemma 9.4.3 on input ϕ to compute a OCP $\mathcal{P}_1 = (D, \delta_0^1, \delta_{>0}^1)$ with initial state $q_I \in D$. The key now is to build on top of \mathcal{P}_1 and the fixed formula α (which can be thought of as an $\text{FO}^2(\text{Reach})$ formula) so as to encode the initial guessing of numbers.

The structure of our output OCP \mathcal{P} can be visualized as

$$B_0 \rightarrow B_1 \dots \rightarrow B_k \rightarrow \mathcal{P}_1.$$

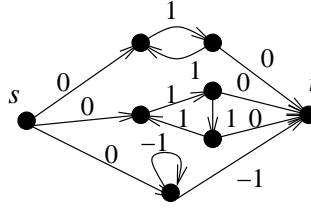


Figure 9.1: The $s \rightarrow^* t$ path-weights in this graph equals $(1 + 2\mathbb{N}) \cup (1 + 3\mathbb{N}) \cup (2 + 3\mathbb{N}) \cup \mathbb{Z}_{<0}$.

The number $k + 1$ of blocks B_i in \mathcal{P} corresponds to the number of rounds played in the buffer game. The initial state is in block B_0 . Our output $\text{FO}^2(\text{Reach})$ formula will have $k + 1$ leading (alternating) quantifiers so as to ensure that each player moves in their designated rounds. One variable will be used for storing the last buffer value from the previous round, while the other is used for storing the buffer value after the designated player has made his move. We now describe how to ensure that at each round i ($i > 0$) the player can only add numbers that are in the set $H_i := \{c \left(\prod_{j=1}^{v_i} p_j \right) : c \in \mathbb{N}\}$. Define the function $g : \mathbb{Z}_{>0} \rightarrow \mathbb{Z}_{>0}$ as $g(s) := \prod_{j=1}^s p_j$. Note that g grows exponentially in s , which is why we cannot simply embed a cycle of length $g(v_i)$ in B_i , for each $i \in [1, k]$. On the other hand, notice that H_i is $\mathbb{Z} - L_i$, where $L_i := \left(\bigcup_{1 \leq j \leq v_i} \bigcup_{a \in (0, p_j)} a + \mathbb{N}p_j \right) \cup \mathbb{Z}_{<0}$.

In turn, L_i can be characterized as the set of weights of paths in a small finite graph G_i from a vertex s to a vertex t , where the *weight* of a path is the sum of the weights of its edges (which we shall allow to be only either -1, 0, or 1). In fact, G_i will have $O(\sum_{j=1}^{v_i} p_j)$ vertices, which is polynomial in v_i . For example, the set $(1 + 2\mathbb{N}) \cup (1 + 3\mathbb{N}) \cup (2 + 3\mathbb{N}) \cup \mathbb{Z}_{<0}$ corresponds to the weights of $s \rightarrow^* t$ paths in the graph in Figure 9.1.

Furthermore, the graph G_i can be thought of as an OCP. Adding the self-loop transitions $(s, \text{loop}_s, s, 0)$ and $(t, \text{loop}_t, t, 0)$ on states s and t , the binary relation

$$\{((s, a), (t, a + b)) : b \in H_i\}$$

can then be expressed in $\text{FO}^2(\text{Reach})$ as $\neg(x \rightarrow^* y) \wedge E_{\text{loop}_s}(x, x) \wedge E_{\text{loop}_t}(y, y)$. Therefore, we shall embed the modified OCP G_i into B_i , where t will be the entry state for block B_{i+1} of \mathcal{P} . [B_{k+1} shall be interpreted as \mathcal{P}_1 .]

Finally, using this idea, it is not difficult to compute the desired $\text{FO}^2(\text{Reach})$ sentence by mimicking the $k + 1$ rounds of the game by using at most $k + c$ alternating quantifiers (using only the variables x and y). The end buffer value m , which needs to

be checked against ϕ , can be checked against α instead. \square

We can also apply Lemma 9.4.1 to prove the following lower bound.

Proposition 9.4.4 *The combined complexity of model checking EF-logic over an asynchronous product of two one-counter processes is PSPACE-hard.*

The proof of this lower bound is given in the appendix. Intuitively, instead of simulating each alternation in the buffer game as values in the two variables x and y , we can simulate them as values in two different counters. We can make sure that the divisibility information is not “overwritten” by encoding it as a non-fixed formula.

We saw in the previous section that the expression complexity of $\text{FO}^2(\text{Reach})$ over ΠOCP is in P . In contrast, we can show that this is not the case for $\text{FO}^4(\text{Reach})$ even over OCPs (without products).

Proposition 9.4.5 *The expression complexity of $\text{FO}^4(\text{Reach})$ (without equality relation) over OCPs is PSPACE-hard.*

The fixed graph is in fact $\langle \mathbb{N}, < \rangle$. The proof, which is given in the appendix, adapts the technique in [GS05] of succinctly encoding addition arithmetic on large numbers using the successor relations and linear order $<$ with only four variables.

We already saw that the data complexity of EF-logic over ΠOCP is P^{NP} . In contrast, we can show the following proposition.

Proposition 9.4.6 *For each $k \in \mathbb{N}$, there is a fixed EF_S -logic formula ϕ_k such that model checking ϕ_k over ΠOCP is Σ_k^P -hard.*

Intuitively, by using the synchronization constraints, one can faithfully simulate two variables x and y in any given $\text{FO}^2(\text{Reach})$ formula as values of two different counters. Again this proof is given in the appendix. This idea can easily be adapted for showing the following proposition by appealing to Proposition 9.4.5.

Proposition 9.4.7 *The expression complexity of EF_S -logic over ΠOCP is hard for PSPACE.*

Part III

Epilogue

Chapter 10

Conclusions and Future work

In this thesis, we have presented several generic and specific techniques for deriving decidability for infinite-state model checking with optimal or near-optimal complexity. This chapter will conclude the thesis with a brief summary of the main results we have obtained in this thesis and some future work.

Summary

We first recapitulate the generic techniques that we have given in this thesis. We adopt word/tree automatic transition systems as our generic framework for modeling infinite-state systems. These classes of systems strike a good balance between expressive power and closure/algorithmic properties. The expressive power of this framework easily yields undecidability even for simple safety properties. Nevertheless, we have obtained several algorithmic metatheorems for showing decidability (with optimal or near-optimal complexity) for various model checking problems over these frameworks. More importantly, we have shown that these algorithmic metatheorems can be used to *uniformly* prove many known or previously not known decidability results with optimal (or near-optimal) complexity. Our algorithmic metatheorems are for recurrent reachability (possibly with generalized Büchi conditions), model checking LTL (or fragments thereof) with multi-regular fairness constraints, and extensions of first-order logic with reachability and extended recurrent reachability operators. Our algorithmic metatheorems can be used to obtain decidability for (among others) pushdown systems, prefix-recognizable systems, regular ground-tree rewrite systems, PA-processes, order-2 collapsible pushdown systems, reversal-bounded counter systems (and their extensions with discrete clocks), and many subclasses of Petri nets. For most of these, we have been able to derive optimal or near-optimal complexity.

We now summarize the specific techniques that we have presented in the second part of this thesis. Most of these techniques are specific for subclasses of counter systems. In particular, we considered reversal-bounded counter systems (and their extensions with discrete clocks) and one-counter processes. For reversal-bounded counter systems (and extensions thereof), we provide a new algorithm for computing Parikh images of NWAs as semilinear sets with optimal complexity, which we then use together with Ibarra's algorithm [Iba78] to obtain an optimal algorithm for computing the reachability relations of such systems. Together with the algorithmic metatheorem from Part I, we obtain an optimal complexity for LTL model checking with multi-regular fairness constraints over reversal-bounded counter systems with discrete clocks. We also provide a kind of fixed-parameter tractability result for model checking EF-logic over reversal-bounded counter systems. For one-counter processes and networks of one-counter processes, we obtain optimal complexity by providing three new fragments of Presburger Arithmetic with better complexity ranging from P^{NP} to PSPACE. We have provided optimal complexity for nearly all model checking problems over these subclasses of counter systems that we considered.

Future work

We close this thesis by several future research directions:

- *Is it possible to develop semi-algorithms for computing reachability relations over word/tree automatic systems with natural and general criteria for completeness?* Recall that reachability for many subclasses of Petri nets and counter systems can be solved by *one* semi-algorithm given in [BFLS05, LS05a] for the class of linear counter systems. In contrast, each of the upper bound that we derive using our algorithmic metatheorems requires the use of a specific known result for computing reachability relations for a specific class of systems. Such a semi-algorithm, if exists, will enable us to perform logic model checking over many classes of infinite-state systems in a *uniform* way.
- *Can our algorithmic metatheorems be extended to ω -automatic transition systems?* As we have mentioned, this class subsumes interesting classes of infinite-state systems including real-timed systems. Furthermore, semi-algorithms for computing a subclass of ω -automatic systems have also been developed [BLW03, BLW09]. However, this problem appears to be rather difficult owing to its con-

nection with the open problem on Ramseyan quantifiers over ω -automatic structures (cf. [Bar07, BGR10, Rub08] for more details).

- *Develop new algorithmic metatheorems with better complexity over Presburger-definable systems.* Our algorithmic metatheorems cannot be used to give optimal complexity for LTL model checking over subclasses of counter systems. For example, LTL model checking over one-counter processes is PSPACE-complete, while our technique only yields EXP upper bound. Restricting to Presburger-definable systems (i.e. a subclass of word automatic systems) could potentially lead to better complexity.
- *Obtain algorithmic metatheorems for branching-time logic model checking with good complexity.* As we saw, the approach that we develop in this thesis can only give nonelementary upper bounds for branching-time logic model checking. This is in contrast to the complexity for pushdown systems or one-counter processes which are within EXP. The challenge is to obtain sufficiently general metatheorems but still with good complexity.
- *Give extensions of the normal form theorem for Parikh images of NWAs, e.g. to one-counter automata.* As saw in Chapter 7, our normal form theorem for Parikh images of NWAs cannot be extended to context-free grammars. There are, however, subclasses of pushdown automata to which our normal form theorem could extend. In particular, an extension to one-counter automata will imply that our exponential time complexity for LTL model checking over reversal-bounded counter systems with discrete clocks also extend to the case when one of the counter in the system is free.

Bibliography

- [Abe95] N. Abe. Characterizing PAC-learnability of semilinear sets. *Inf. Comput.*, 116(1):81–102, 1995.
- [ABS01] A. Annichini, A. Bouajjani, and M. Sighireanu. TReX: A tool for reachability analysis of complex systems. In G. Berry, H. Comon, and A. Finkel, editors, *CAV*, volume 2102 of *Lecture Notes in Computer Science*, pages 368–372. Springer, 2001.
- [ACJT96] P. A. Abdulla, K. Cerans, B. Jonsson, and Y. K. Tsay. General decidability theorems for infinite-state systems. In *LICS*, pages 313–321, 1996.
- [AD94] R. Alur and D. L. Dill. A theory of timed automata. *Theor. Comput. Sci.*, 126(2):183–235, 1994.
- [AHU83] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *Data Structures and Algorithms*. Addison-Wesley, 1983.
- [AHV95] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.
- [AJNS04] P. A. Abdulla, B. Jonsson, M. Nilsson, and M. Saksena. A survey of regular model checking. In *Proc. of CONCUR04*, volume 3170 of *LNCS*, pages 35–48. Springer, 2004.
- [AJRS06] P. A. Abdulla, B. Jonsson, A. Rezine, and M. Saksena. Proving liveness by backwards reachability. In C. Baier and H. Hermanns, editors, *CONCUR*, volume 4137 of *Lecture Notes in Computer Science*, pages 95–109. Springer, 2006.
- [All01] E. Allender. The division breakthroughs. *Bulletin of the EATCS*, 74:61–77, 2001.

- [AM04] R. Alur and P. Madhusudan. Decision problems for timed automata: A survey. In M. Bernardo and F. Corradini, editors, *SFM*, volume 3185 of *Lecture Notes in Computer Science*, pages 1–24. Springer, 2004.
- [AP04] R. Alur and D. Peled, editors. *Computer Aided Verification, 16th International Conference, CAV 2004, Boston, MA, USA, July 13-17, 2004, Proceedings*, volume 3114 of *Lecture Notes in Computer Science*. Springer, 2004.
- [BA06] M. Ben-Ari. *Principles of Concurrent and Distributed Programming*. Addison Wesley, second edition, 2006.
- [Bar07] V. Barany. *Automatic Presentations of Infinite Structures*. PhD thesis, RWTH Aachen, 2007.
- [BBF⁺01] B. Bérard, M. Bidoit, A. Finkel, F. Laroussinie, A. Petit, L. Petrucci, and Ph. Schnoebelen. *Systems and Software Verification: Model-Checking Techniques and Tools*. Springer, 2001.
- [BC96] A. Boudet and H. Comon. Diophantine equations, Presburger arithmetic and finite automata. In H. Kirchner, editor, *CAAP*, volume 1059 of *Lecture Notes in Computer Science*, pages 30–43. Springer, 1996.
- [BCM⁺90] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang. Symbolic model checking: 10^{20} states and beyond. In Meyer [Mey90], pages 428–439.
- [BCMS01] O. Burkart, D. Caucal, F. Moller, and B. Steffen. Verification on infinite structures. In *Handbook of process algebra*, pages 545–623. Elsevier, North-Holland, 2001. Chapter 9.
- [BDEK07] B. Becker, C. Dax, J. Eisinger, and F. Klaedtke. LIRA: Handling constraints of linear arithmetics over the integers and the reals. In *Proceedings of the 19th International Conference on Computer Aided Verification (CAV’07)*, volume 4590 of *Lecture Notes in Computer Science*, pages 307–310. Springer-Verlag, 2007.
- [BdRV01] P. Blackburn, M. de Rijke, and Y. Venema. *Modal Logic*. Cambridge University Press, 2001.

- [BEM97] A. Bouajjani, J. Esparza, and O. Maler. Reachability analysis of push-down automata: Application to model-checking. In A. W. Mazurkiewicz and J. Winkowski, editors, *CONCUR*, volume 1243 of *Lecture Notes in Computer Science*, pages 135–150. Springer, 1997.
- [Ber79] J. Berstel. *Transductions and Context-free Languages*. Teubner Verlag, 1979.
- [Ber80] L. Berman. The complexity of logical theories. *Theor. Comput. Sci.*, 11:71–77, 1980.
- [BFLP08] S. Bardin, A. Finkel, J. Leroux, and L. Petrucci. FAST: acceleration from theory to practice. *STTT*, 10(5):401–424, 2008.
- [BFLS05] S. Bardin, A. Finkel, J. Leroux, and Ph. Schnoebelen. Flat acceleration in symbolic model checking. In Peled and Tsay [PT05], pages 474–488.
- [BG04] A. Blumensath and E. Grädel. Finite presentations of infinite structures: Automata and interpretations. *Theory Comput. Syst.*, 37(6):641–674, 2004.
- [BG09] W. Bekker and V. Goranko. Symbolic model checking of tense logics on rational Kripke models. *LNAI*, 5489:3–21, 2009.
- [BGR10] V. Barany, E. Graedel, and S. Rubin. Automata-based presentations of infinite structures, 2010.
- [BH96] A. Bouajjani and P. Habermehl. Constrained properties, semilinear systems, and Petri nets. In Montanari and Sassone [MS96], pages 481–497.
- [BHLW10] P. Barceló, C. A. Hurtado, L. Libkin, and P. T. Wood. Expressive languages for path queries over graph-structured data. In J. Paredaens and D. Van Gucht, editors, *PODS*, pages 3–14. ACM, 2010.
- [BHMV94] V. Bruyère, G. Hansel, C. Michaux, and R. Villemaire. Logic and p -recognizable sets of integers. *Bull. Belg. Math. Soc.*, 1:191–238, 1994.
- [BHV04] A. Bouajjani, P. Habermehl, and T. Vojnar. Abstract regular model checking. In Alur and Peled [AP04], pages 372–386.

- [Bir92] J. C. Birget. Intersection and union of regular languages and state complexity. *Inf. Process. Lett.*, 43(4):185–190, 1992.
- [BJNT00] A. Bouajjani, B. Jonsson, M. Nilsson, and T. Touili. Regular model checking. In Emerson and Sistla [ES00], pages 403–418.
- [BKRS09] L. Bozzelli, M. Kretínský, V. Reháč, and J. Strejcek. On decidability of LTL model checking for process rewrite systems. *Acta Inf.*, 46(1):1–28, 2009.
- [BLN07] M. Benedikt, L. Libkin, and F. Neven. Logical definability and query languages over ranked and unranked trees. *ACM Trans. Comput. Log.*, 8(2), 2007.
- [Blu99] A. Blumensath. Automatic structures. Master’s thesis, RWTH Aachen, 1999.
- [BLW03] B. Boigelot, A. Legay, and P. Wolper. Iterating transducers in the large (extended abstract). In W. A. Hunt Jr. and F. Somenzi, editors, *CAV*, volume 2725 of *Lecture Notes in Computer Science*, pages 223–235. Springer, 2003.
- [BLW09] A. Bouajjani, A. Legay, and P. Wolper. A framework to handle linear temporal properties in (ω)-regular model checking. *CoRR*, abs/0901.4080, 2009.
- [BMMR01] T. Ball, R. Majumdar, T. Millstein, and S. K. Rajamani. Automatic predicate abstraction of C programs. *SIGPLAN Not.*, 36(5):203–213, 2001.
- [Boi99] B. Boigelot. *Symbolic Methods for Exploring Infinite State Spaces*. PhD thesis, Université de Liège, 1999.
- [Boi03] B. Boigelot. On iterating linear transformations over recognizable sets of integers. *Theor. Comput. Sci.*, 309(1-3):413–468, 2003.
- [Bou01] A. Bouajjani. Languages, rewriting systems, and verification of infinite-state systems. In F. Orejas, P. G. Spirakis, and J. van Leeuwen, editors, *ICALP*, volume 2076 of *Lecture Notes in Computer Science*, pages 24–39. Springer, 2001.

- [BŘS09] T. Babiak, V. Řehák, and J. Strejček. Almost Linear Büchi Automata. In *Proceedings 16th International Workshop on Expressiveness in Concurrency 2009 (EXPRESS'09). EPTCS 8*, pages 16–25, 2009.
- [BW90] J. Baeten and W. Weijland. *Process Algebra*. Cambridge University Press, 1990.
- [BW94] B. Boigelot and P. Wolper. Symbolic verification with periodic sets. In D. L. Dill, editor, *CAV*, volume 818 of *Lecture Notes in Computer Science*, pages 55–67. Springer, 1994.
- [Cac02] T. Cachat. Uniform solution of parity games on prefix-recognizable graphs. *Electr. Notes Theor. Comput. Sci.*, 68(6), 2002.
- [Cac03] T. Cachat. Higher order pushdown automata, the Caucal hierarchy of graphs and parity games. In J. C. M. Baeten, J. K. Lenstra, J. Parrow, and G. J. Woeginger, editors, *ICALP*, volume 2719 of *Lecture Notes in Computer Science*, pages 556–569. Springer, 2003.
- [Cau90] D. Caucal. On the regular structure of prefix rewriting. In A. Arnold, editor, *CAAP*, volume 431 of *Lecture Notes in Computer Science*, pages 87–102. Springer, 1990.
- [Cau92] D. Caucal. On the regular structure of prefix rewriting. *Theor. Comput. Sci.*, 106(1):61–86, 1992.
- [Cau96] D. Caucal. On infinite transition graphs having a decidable monadic theory. In F. Meyer auf der Heide and B. Monien, editors, *ICALP*, volume 1099 of *Lecture Notes in Computer Science*, pages 194–205. Springer, 1996.
- [Cau02] D. Caucal. On infinite terms having a decidable monadic theory. In Diks and Rytter [DR02], pages 165–176.
- [Cau03] D. Caucal. On infinite transition graphs having a decidable monadic theory. *Theor. Comput. Sci.*, 290(1):79–115, 2003.
- [CDG⁺07] H. Comon, M. Dauchet, R. Gilleron, C. Löding, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications. Available on: <http://www.grappa.univ-lille3.fr/tata>, 2007. release October, 12th 2007.

- [CDGV94] J. L. 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.
- [CGP99] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model checking*. The MIT Press, 1999.
- [CH90] K. J. Compton and C. W. Henson. A uniform method for proving lower bounds on the computational complexity of logical theories. *Ann. Pure Appl. Logic*, 48(1):1–79, 1990.
- [Chr86] M. Chrobak. Finite automata and unary languages. *Theor. Comput. Sci.*, 47(3):149–158, 1986.
- [Cip95] B. Cipra. How number theory got the best of the Pentium chip. *Science*, 267(5195):175, 1995.
- [CJ98] H. Comon and Y. Jurski. Multiple counters automata, safety analysis and Presburger arithmetic. In Hu and Vardi [HV98], pages 268–279.
- [CJ99] H. Comon and Y. Jurski. Timed automata and the theory of real numbers. In J. C. M. Baeten and S. Mauw, editors, *CONCUR*, volume 1664 of *Lecture Notes in Computer Science*, pages 242–257. Springer, 1999.
- [Col02] T. Colcombet. On families of graphs having a decidable first order theory with reachability. In P. Widmayer, F. T. Ruiz, R. M. Bueno, M. Hennessy, S. Eidenbenz, and R. Conejo, editors, *ICALP*, volume 2380 of *Lecture Notes in Computer Science*, pages 98–109. Springer, 2002.
- [CPR06a] B. Cook, A. Podelski, and A. Rybalchenko. Termination proofs for systems code. In M. I. Schwartzbach and T. Ball, editors, *PLDI*, pages 415–426. ACM, 2006.
- [CPR06b] B. Cook, A. Podelski, and A. Rybalchenko. Terminator: Beyond safety. In T. Ball and R. B. Jones, editors, *CAV*, volume 4144 of *Lecture Notes in Computer Science*, pages 415–418. Springer, 2006.
- [CW98] B. Courcelle and I. Walukiewicz. Monadic second-order logic, graph coverings and unfoldings of transition systems. *Ann. Pure Appl. Logic*, 92(1):35–62, 1998.

- [CW03] A. Carayol and S. Wöhrle. The Caucal hierarchy of infinite graphs in terms of logic and higher-order pushdown automata. In P. K. Pandya and J. Radhakrishnan, editors, *FSTTCS*, volume 2914 of *Lecture Notes in Computer Science*, pages 112–123. Springer, 2003.
- [CW07] T. Cachat and I. Walukiewicz. The complexity of games on higher order pushdown automata. *CoRR*, abs/0705.0262, 2007.
- [CY92] C. Courcoubetis and M. Yannakakis. Minimum and maximum delay problems in real-time systems. *Formal Methods in System Design*, 1(4):385–415, 1992.
- [Dem06] S. Demri. Linear-time temporal logics with Presburger constraints: An overview. *Journal of Applied Non-Classical Logics*, 16(3-4):311–347, 2006.
- [DGH] A. Durand-Gasselin and P. Habermehl. On the use of nondeterministic automata for Presburger arithmetic. To appear in CONCUR 2010.
- [DIB⁺00] Z. Dang, O. H. Ibarra, T. Bultan, R. A. Kemmerer, and J. Su. Binary reachability analysis of discrete pushdown timed automata. In Emerson and Sistla [ES00], pages 69–84.
- [Dic13] L.E. Dickson. Finiteness of the odd perfect and primitive abundant numbers with r distinct prime factors. *Amer. Journal Math.*, 35:413–422, 1913.
- [DIP01] Z. Dang, O. H. Ibarra, and P. S. Pietro. Liveness verification of reversal-bounded multicounter machines with a free counter. In R. Hariharan, M. Mukund, and V. Vinay, editors, *FSTTCS*, volume 2245 of *Lecture Notes in Computer Science*, pages 132–143. Springer, 2001.
- [DKW08] V. D’Silva, D. Kroening, and G. Weissenbacher. A survey of automated techniques for formal software verification. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 27(7):1165–1178, 2008.
- [DLS02] D. Dams, Y. Lakhnech, and M. Steffen. Iterating transducers. *J. Log. Algebr. Program.*, 52-53:109–127, 2002.

- [DR02] K. Diks and W. Rytter, editors. *Mathematical Foundations of Computer Science 2002, 27th International Symposium, MFCS 2002, Warsaw, Poland, August 26-30, 2002, Proceedings*, volume 2420 of *Lecture Notes in Computer Science*. Springer, 2002.
- [DT90] M. Dauchet and S. Tison. The theory of ground rewrite systems is decidable. In Meyer [Mey90], pages 242–248.
- [Edm67] J. Edmonds. Systems of distinct representatives and linear algebra. *J. Res. Nat. Bur. Standards*, 71B:241–245, 1967.
- [EHRS00] J. Esparza, D. Hansel, P. Rossmanith, and S. Schwoon. Efficient algorithms for model checking pushdown systems. In Emerson and Sistla [ES00], pages 232–247.
- [EK99] J. Esparza and J. Knoop. An automata-theoretic approach to interprocedural data-flow analysis. In W. Thomas, editor, *FoSSaCS*, volume 1578 of *Lecture Notes in Computer Science*, pages 14–30. Springer, 1999.
- [EKS03] J. Esparza, A. Kucera, and S. Schwoon. Model checking LTL with regular valuations for pushdown systems. *Inf. Comput.*, 186(2):355–376, 2003.
- [EP00] J. Esparza and A. Podelski. Efficient algorithms for pre^* and post^* on interprocedural parallel flow graphs. In *POPL*, pages 1–11, 2000.
- [ES00] E. A. Emerson and A. P. Sistla, editors. *Computer Aided Verification, 12th International Conference, CAV 2000, Chicago, IL, USA, July 15-19, 2000, Proceedings*, volume 1855 of *Lecture Notes in Computer Science*. Springer, 2000.
- [Esp94] J. Esparza. On the decidability of model checking for several μ -calculi and petri nets. In S. Tison, editor, *CAAP*, volume 787 of *Lecture Notes in Computer Science*, pages 115–129. Springer, 1994.
- [Esp96] J. Esparza. Decidability and complexity of Petri net problems - an introduction. In W. Reisig and G. Rozenberg, editors, *Petri Nets*, volume 1491 of *Lecture Notes in Computer Science*, pages 374–428. Springer, 1996.
- [Esp97a] J. Esparza. Decidability of model checking for infinite-state concurrent systems. *Acta Inf.*, 34(2):85–107, 1997.

- [Esp97b] J. Esparza. Petri nets, commutative context-free grammars, and basic parallel processes. *Fundam. Inf.*, 31(1):13–25, 1997.
- [EVW02] K. Etessami, M. Y. Vardi, and T. Wilke. First-order logic with two variables and unary temporal logic. *Inf. Comput.*, 179(2):279–295, 2002.
- [FG06] J. Flum and M. Grohe. *Parameterized Complexity Theory*. Springer, 2006.
- [Fin87] A. Finkel. A generalization of the procedure of Karp and Miller to well structured transition systems. In T. Ottmann, editor, *ICALP*, volume 267 of *Lecture Notes in Computer Science*, pages 499–508. Springer, 1987.
- [Fin90] A. Finkel. Reduction and covering of infinite reachability trees. *Inf. Comput.*, 89(2):144–179, 1990.
- [FL02] A. Finkel and J. Leroux. How to compose Presburger-accelerations: Applications to broadcast protocols. In M. Agrawal and A. Seth, editors, *FSTTCS*, volume 2556 of *Lecture Notes in Computer Science*, pages 145–156. Springer, 2002.
- [FS01] A. Finkel and Ph. Schnoebelen. Well-structured transition systems everywhere! *Theor. Comput. Sci.*, 256(1-2):63–92, 2001.
- [GI81] E. M. Gurari and O. H. Ibarra. The complexity of decision problems for finite-turn multicounter machines. *J. Comput. Syst. Sci.*, 22(2):220–229, 1981.
- [GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness*. W.H.Freeman & Co Ltd, 1979.
- [GL06] S. Göller and M. Lohrey. Infinite state model-checking of propositional dynamic logics. In Z. Ésik, editor, *CSL*, volume 4207 of *Lecture Notes in Computer Science*, pages 349–364. Springer, 2006.
- [GL10] S. Göller and M. Lohrey. Branching-time model checking of one-counter processes. In Marion and Schwentick [MS10], pages 405–416.
- [GMT09] S. Göller, R. Mayr, and A. W. To. On the computational complexity of verifying one-counter processes. In *LICS*, pages 235–244. IEEE Computer Society, 2009.

- [GN08] W. Gelade and F. Neven. Succinctness of the complement and intersection of regular expressions. In S. Albers and P. Weil, editors, *STACS*, volume 1 of *LIPICs*, pages 325–336. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany, 2008.
- [Göl08] S. Göller. Reachability on prefix-recognizable graphs. *Inf. Process. Lett.*, 108(2):71–74, 2008.
- [Grä88] E. Grädel. Subclasses of Presburger arithmetic and the polynomial-time hierarchy. *Theor. Comput. Sci.*, 56:289–301, 1988.
- [Grä90] E. Grädel. Simple interpretations among complicated theories. *Inf. Process. Lett.*, 35(5):235–238, 1990.
- [Gre09] S. Greengard. Making automation work. *Commun. ACM*, 52(12):18–19, 2009.
- [Gru97] O. Grumberg, editor. *Computer Aided Verification, 9th International Conference, CAV '97, Haifa, Israel, June 22-25, 1997, Proceedings*, volume 1254 of *Lecture Notes in Computer Science*. Springer, 1997.
- [GS66] S. Ginsburg and E. H. Spanier. Semigroups, Presburger formulas, and languages. *Pacific J. Math.*, 16(2):285–296, 1966.
- [GS78] J. Von Zur Gathen and M. Sieveking. A bound on solutions of linear integer equalities and inequalities. *Proc. AMS*, 72:155–158, 1978.
- [GS97] S. Graf and H. Saïdi. Construction of abstract state graphs with PVS. In Grumberg [Gru97], pages 72–83.
- [GS00] S. Graf and M. I. Schwartzbach, editors. *Tools and Algorithms for Construction and Analysis of Systems, 6th International Conference, TACAS 2000, Held as Part of the European Joint Conferences on the Theory and Practice of Software, ETAPS 2000, Berlin, Germany, March 25 - April 2, 2000, Proceedings*, volume 1785 of *Lecture Notes in Computer Science*. Springer, 2000.
- [GS05] M. Grohe and N. Schweikardt. The succinctness of first-order logic on linear orders. *Logical Methods in Computer Science*, 1(1), 2005.

- [Hab97] P. Habermehl. On the complexity of the linear-time mu-calculus for Petri-nets. In P. Azéma and G. Balbo, editors, *ICATPN*, volume 1248 of *Lecture Notes in Computer Science*, pages 102–116. Springer, 1997.
- [Har86] D. Harel. Effective transformations on infinite trees, with applications to high undecidability, dominoes, and fairness. *J. ACM*, 33(1):224–248, 1986.
- [HJJ⁺95] J.G. Henriksen, J. Jensen, M. Jørgensen, N. Klarlund, B. Paige, T. Rauhe, and A. Sandholm. Mona: Monadic second-order logic in practice. In *Tools and Algorithms for the Construction and Analysis of Systems, First International Workshop, TACAS '95, LNCS 1019*, 1995.
- [HMOS08] M. Hague, A. S. Murawski, C.-H. L. Ong, and O. Serre. Collapsible pushdown automata and recursion schemes. In *LICS*, pages 452–461. IEEE Computer Society, 2008.
- [Hod83] B. R. Hodgson. Decidabilité par automate fini. *Ann. Sc. Math. Quebec*, 7(1):39–57, 1983.
- [HP79] J. E. Hopcroft and J. J. Pansiot. On the reachability problem for 5-dimensional vector addition systems. *Theor. Comput. Sci.*, 8:135–159, 1979.
- [HPW09a] C. A. Hurtado, A. Poulouvasilis, and P. T. Wood. Finding top-*k* approximate answers to path queries. In T. Andreasen, R. R. Yager, H. Bulskov, H. Christiansen, and H. L. Larsen, editors, *FQAS*, volume 5822 of *Lecture Notes in Computer Science*, pages 465–476. Springer, 2009.
- [HPW09b] C. A. Hurtado, A. Poulouvasilis, and P. T. Wood. Ranking approximate answers to semantic web queries. In L. Aroyo, P. Traverso, F. Ciravegna, P. Cimiano, T. Heath, E. Hyvönen, R. Mizoguchi, E. Oren, M. Sabou, and E. P. B. Simperl, editors, *ESWC*, volume 5554 of *Lecture Notes in Computer Science*, pages 263–277. Springer, 2009.
- [HR87] R. R. Howell and L. E. Rosier. An analysis of the nonemptiness problem for classes of reversal-bounded multicounter machines. *J. Comput. Syst. Sci.*, 34(1):55–74, 1987.

- [Huy83] D. T. Huynh. Commutative grammars: The complexity of uniform word problems. *Information and Control*, 57(1):21–39, 1983.
- [HV98] A. J. Hu and M. Y. Vardi, editors. *Computer Aided Verification, 10th International Conference, CAV '98, Vancouver, BC, Canada, June 28 - July 2, 1998, Proceedings*, volume 1427 of *Lecture Notes in Computer Science*. Springer, 1998.
- [Iba78] O. H. Ibarra. Reversal-bounded multicounter machines and their decision problems. *J. ACM*, 25(1):116–133, 1978.
- [IK89] N. Immerman and D. Kozen. Definability with bounded number of bound variables. *Inf. Comput.*, 83(2):121–139, 1989.
- [ISD⁺02] O. H. Ibarra, J. Su, Z. Dang, T. Bultan, and R. A. Kemmerer. Counter machines and verification problems. *Theor. Comput. Sci.*, 289(1):165–189, 2002.
- [JKM01] P. Jancar, A. Kucera, and R. Mayr. Deciding bisimulation-like equivalences with finite-state processes. *Theor. Comput. Sci.*, 258(1-2):409–433, 2001.
- [JKMS04] P. Jancar, A. Kucera, F. Moller, and Z. Sawa. DP lower bounds for equivalence-checking and model-checking of one-counter automata. *Inf. Comput.*, 188(1):1–19, 2004.
- [JM00] D. Jurafsky and J. H. Martin. *Speech and Language Processing*. Prentice Hall, 2000.
- [JN00] B. Jonsson and M. Nilsson. Transitive closures of regular relations for verifying infinite-state systems. In Graf and Schwartzbach [GS00], pages 220–234.
- [Joh86] J. H. Johnson. Rational equivalence relations. *Theor. Comput. Sci.*, 47(3):39–60, 1986.
- [JW96] D. Janin and I. Walukiewicz. On the expressive completeness of the propositional mu-calculus with respect to monadic second order logic. In Montanari and Sassone [MS96], pages 263–277.

- [Kam68] H. Kamp. *Tense Logic and the Theory of Linear Order*. PhD thesis, UCLA, 1968.
- [Kan83] R. Kannan. Improved algorithms for integer programming and related lattice problems. In *STOC*, pages 193–206. ACM, 1983.
- [Kar10] A. Kartzow. Collapsible pushdown graphs of level 2 are tree-automatic. In Marion and Schwentick [MS10], pages 501–512.
- [KEM06] A. Kucera, J. Esparza, and R. Mayr. Model checking probabilistic pushdown automata. *Logical Methods in Computer Science*, 2(1), 2006.
- [KJ06] A. Kucera and P. Jancar. Equivalence-checking on infinite-state systems: Techniques and results. *TPLP*, 6(3):227–264, 2006.
- [Kla08] F. Klaedtke. Bounds on the automata size for Presburger arithmetic. *ACM Trans. Comput. Log.*, 9(2), 2008.
- [KM02] A. Kucera and R. Mayr. On the complexity of semantic equivalences for pushdown automata and BPA. In Diks and Rytter [DR02], pages 433–445.
- [KMM⁺97] Y. Kesten, O. Maler, M. Marcus, A. Pnueli, and E. Shahar. Symbolic model checking with rich ssertional languages. In Grumberg [Gru97], pages 424–435.
- [KMM⁺01] Y. Kesten, O. Maler, M. Marcus, A. Pnueli, and E. Shahar. Symbolic model checking with rich assertional languages. *Theor. Comput. Sci.*, 256(1-2):93–112, 2001.
- [Kop10] E. Kopczynski. Complexity of problems for commutative grammars. *CoRR*, abs/1003.4105, 2010.
- [Koz97] D. C. Kozen. *Automata and Computability*. Springer-Verlag, 1997.
- [Koz06] D. C. Kozen. *Theory of Computation*. Springer-Verlag, 2006.
- [KPV02] O. Kupferman, N. Piterman, and M. Y. Vardi. Model checking linear properties of prefix-recognizable systems. In E. Brinksma and K. G. Larsen, editors, *CAV*, volume 2404 of *Lecture Notes in Computer Science*, pages 371–385. Springer, 2002.

- [KRS05] B. Khoussainov, S. Rubin, and F. Stephan. Automatic linear orders and trees. *ACM Trans. Comput. Log.*, 6(4):675–700, 2005.
- [KT10] E. Kopczynski and A. W. To. Parikh images of grammars: Complexity and applications. *To appear in LICS*, 2010.
- [Kuc00] A. Kucera. Efficient verification algorithms for one-counter processes. In U. Montanari, J. D. P. Rolim, and Emo Welzl, editors, *ICALP*, volume 1853 of *Lecture Notes in Computer Science*, pages 317–328. Springer, 2000.
- [LAS] Homepage LASH. <http://www.montefiore.ulg.ac.be/~boigelot/research/lash/>.
- [Len83] H.W. Lenstra. Integer programming with a fixed number of variables. *Math. Oper. Res.*, 8(4):538–548, 1983.
- [Lib04] L. Libkin. *Elements Of Finite Model Theory (Texts in Theoretical Computer Science. An EATCS Series)*. Springer-Verlag, 2004.
- [Lip76] R. J. Lipton. The reachability problem requires exponential space. Technical Report 62, Department of Computer Science, Yale University, 1976.
- [Löd03] C. Löding. *Infinite Graphs Generated by Tree Rewriting*. PhD thesis, RWTH Aachen, 2003.
- [Löd06] C. Löding. Reachability problems on regular ground tree rewriting graphs. *Theory Comput. Syst.*, 39(2):347–383, 2006.
- [LS02] D. Lugiez and Ph. Schnoebelen. The regular viewpoint on PA-processes. *Theor. Comput. Sci.*, 274(1-2):89–115, 2002.
- [LS04] J. Leroux and G. Sutre. On flatness for 2-dimensional vector addition systems with states. In P. Gardner and N. Yoshida, editors, *CONCUR*, volume 3170 of *Lecture Notes in Computer Science*, pages 402–416. Springer, 2004.
- [LS05a] J. Leroux and G. Sutre. Flat counter automata almost everywhere! In Peled and Tsay [PT05], pages 489–503.

- [LS05b] D. Lugiez and Ph. Schnoebelen. Decidable first-order transition logics for PA-processes. *Inf. Comput.*, 203(1):75–113, 2005.
- [LS07] M. Leucker and C. Sánchez. Regular linear temporal logic. In C. B. Jones, Z. Liu, and J. Woodcock, editors, *ICTAC*, volume 4711 of *Lecture Notes in Computer Science*, pages 291–305. Springer, 2007.
- [Lyn96] N. A. Lynch. *Distributed Algorithms*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1996.
- [Mai00] M. Maidl. The common fragment of CTL and LTL. In *FOCS*, pages 643–652, 2000.
- [Mak04] J. A. Makowsky. Algorithmic uses of the Feferman-Vaught theorem. *Ann. Pure Appl. Logic*, 126(1-3):159–213, 2004.
- [Mar02] A. Martinez. Efficient computation of regular expressions from unary NFAs. In *DFCS '02: Pre-Proceedings, Descriptive Complexity of Formal Systems*, pages 174–187, 2002.
- [Mas76] A.N. Maslov. Multilevel stack automata. *Probl. Inf. Transm.*, 15:1170–1174, 1976.
- [May84] E. W. Mayr. An algorithm for the general Petri net reachability problem. *SIAM J. Comput.*, 13(3):441–460, 1984.
- [May98] R. Mayr. *Decidability and Complexity of Model Checking Problems for Infinite-State Systems*. PhD thesis, TU-Munich, 1998.
- [May00] R. Mayr. On the complexity of bisimulation problems for pushdown automata. In J. van Leeuwen, O. Watanabe, M. Hagiya, P. D. Mosses, and T. Ito, editors, *IFIP TCS*, volume 1872 of *Lecture Notes in Computer Science*, pages 474–488. Springer, 2000.
- [May01] R. Mayr. Decidability of model checking with the temporal logic EF. *Theor. Comput. Sci.*, 256(1-2):31–62, 2001.
- [McM93] K. L. McMillan. *Symbolic Model Checking: An Approach to the State Explosion Problem*. Kluwer Academic Publishers, 1993.

- [Mey90] A. R. Meyer, editor. *Proceedings, Fifth Annual IEEE Symposium on Logic in Computer Science, 4-7 June 1990, Philadelphia, Pennsylvania, USA*. IEEE Computer Society, 1990.
- [Min67] M. L. Minsky. *Computation: Finite and Infinite Machines*. Prentice-Hall, Englewood Cliff, NJ, 1967.
- [Mol96] F. Moller. Infinite results. In Montanari and Sassone [MS96], pages 195–216.
- [Mor00] C. Morvan. On rational graphs. In *FOSSACS '00*, pages 252–266, 2000.
- [MP71] M. McNaughton and S. Papert. *Counter-Free Automata*. MIT Press, 1971.
- [MS85] D. E. Muller and P. E. Schupp. The theory of ends, pushdown automata, and second-order logic. *Theor. Comput. Sci.*, 37:51–75, 1985.
- [MS96] U. Montanari and V. Sassone, editors. *CONCUR '96, Concurrency Theory, 7th International Conference, Pisa, Italy, August 26-29, 1996, Proceedings*, volume 1119 of *Lecture Notes in Computer Science*. Springer, 1996.
- [MS10] J.-Y. Marion and T. Schwentick, editors. *27th International Symposium on Theoretical Aspects of Computer Science, STACS 2010, March 4-6, 2010, Nancy, France*, volume 5 of *LIPICs*. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2010.
- [MW95] A. O. Mendelzon and P. T. Wood. Finding regular simple paths in graph databases. *SIAM J. Comput.*, 24(6):1235–1258, 1995.
- [Nil05] M. Nilsson. *Regular Model Checking*. PhD thesis, Uppsala Universitet, 2005.
- [Ome] Homepage Omega. <http://www.cs.umd.edu/projects/omega/>.
- [Ong06] C.-H. L. Ong. On model-checking trees generated by higher-order recursion schemes. In *LICS*, pages 81–90. IEEE Computer Society, 2006.
- [Pap81] C. H. Papadimitriou. On the complexity of integer programming. *J. ACM*, 28(4):765–768, 1981.

- [Pap94] C. H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.
- [Par66] R. J. Parikh. On context-free languages. *J. ACM*, 13(4):570–581, 1966.
- [Pél92] P. Péladeau. Logically defined subsets of \mathbb{N}^k . *Theor. Comput. Sci.*, 93(2):169–183, 1992.
- [PS98] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Dover Publications, 1998.
- [PSW02] G. Pighizzini, J. Shallit, and M.-W. Wang. Unary context-free grammars and pushdown automata, descriptive complexity and auxiliary space lower bounds. *J. Comput. Syst. Sci.*, 65(2):393–414, 2002.
- [PT05] D. Peled and Y.-K. Tsay, editors. *Automated Technology for Verification and Analysis, Third International Symposium, ATVA 2005, Taipei, Taiwan, October 4-7, 2005, Proceedings*, volume 3707 of *Lecture Notes in Computer Science*. Springer, 2005.
- [PV04] N. Piterman and M. Y. Vardi. Global model-checking of infinite-state systems. In Alur and Peled [AP04], pages 387–400.
- [Rab70] M. O. Rabin. Weakly definable relations and special automata. In Y. Bar-Hillel, editor, *Proc. Symp. Math. Logic and Foundations of Set Theory*, pages 1–23. North Holland, 1970.
- [Rab07] A. Rabinovich. On compositionality and its limitations. *ACM Trans. Comput. Log.*, 8(1), 2007.
- [Reh07] V. Rehak. *On Extensions of Process Rewrite Systems*. PhD thesis, Masaryk University, 2007.
- [Rub08] S. Rubin. Automata presenting structures: A survey of the finite string case. *Bulletin of Symbolic Logic*, 14(2):169–209, 2008.
- [SC85] A. P. Sistla and E. M. Clarke. The complexity of propositional linear temporal logics. *J. ACM*, 32(3):733–749, 1985.
- [Sca84] B. Scarpellini. Complexity of subcases of Presburger arithmetic. *Transactions of the American Mathematical Society*, 284(1):203–218, 1984.

- [Sch02] Ph. Schnoebelen. The complexity of temporal logic model checking. In P. Balbiani, N.-Y. Suzuki, F. Wolter, and M. Zakharyashev, editors, *Advances in Modal Logic*, pages 393–436. King’s College Publications, 2002.
- [Sem84] A. L. Semenov. Decidability of monadic theories. In M. Chytil and V. Koubek, editors, *MFCS*, volume 176 of *Lecture Notes in Computer Science*, pages 162–175. Springer, 1984.
- [Ser06] O. Serre. Parity games played on transition graphs of one-counter processes. In L. Aceto and A. Ingólfssdóttir, editors, *FoSSaCS*, volume 3921 of *Lecture Notes in Computer Science*, pages 337–351. Springer, 2006.
- [Sip97] M. Sipser. *Introduction to the Theory of Computation*. PWS Publishing Company, 1997.
- [SL10] C. Sánchez and M. Leucker. Regular linear temporal logic with past. In G. Barthe and M. V. Hermenegildo, editors, *VMCAI*, volume 5944 of *Lecture Notes in Computer Science*, pages 295–311. Springer, 2010.
- [SM73] L. J. Stockmeyer and A. R. Meyer. Word problems requiring exponential time: Preliminary report. In *STOC*, pages 1–9. ACM, 1973.
- [SSM07] H. Seidl, Th. Schwentick, and A. Muscholl. Counting in trees. *Texts in Logic and Games*, 2:575–612, 2007.
- [SSMH04] H. Seidl, T. Schwentick, A. Muscholl, and P. Habermehl. Counting in trees for free. In J. Díaz, J. Karhumäki, A. Lepistö, and D. Sannella, editors, *ICALP*, volume 3142 of *Lecture Notes in Computer Science*, pages 1136–1149. Springer, 2004.
- [Sti98] C. Stirling. The joys of bisimulation. In L. Brim, J. Gruska, and J. Zlatuska, editors, *MFCS*, volume 1450 of *Lecture Notes in Computer Science*, pages 142–151. Springer, 1998.
- [Sti01] C. Stirling. *Modal and temporal properties of processes*. Springer-Verlag New York, Inc., New York, NY, USA, 2001.
- [Sto74] L. J. Stockmeyer. *The complexity of decision problems in automata theory and logic*. PhD thesis, Department of Electrical Engineering, MIT, 1974.

- [SV00] H. Spakowski and J. Vogel. Theta2p-completeness: A classical approach for new results. In *Proc. of FST&TCS 2000*, volume 1974 of *LNCS*, pages 348–360. Springer, 2000.
- [Tho96] W. Thomas. Languages, automata, and logic. In *Handbook of Formal Languages*, pages 389–455. Springer, 1996.
- [Tho03] W. Thomas. Constructing infinite graphs with a decidable MSO-theory. In B. Rován and P. Vojtás, editors, *MFCS*, volume 2747 of *Lecture Notes in Computer Science*, pages 113–124. Springer, 2003.
- [TL08] A. W. To and L. Libkin. Recurrent reachability analysis in regular model checking. In *LPAR '08: Proceedings of the 15th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning*, pages 198–213, Berlin, Heidelberg, 2008. Springer-Verlag.
- [TL10] A. W. To and L. Libkin. Algorithmic metatheorems for decidable LTL model checking over infinite systems. In C.-H. Luke Ong, editor, *FOS-SACS*, volume 6014 of *Lecture Notes in Computer Science*, pages 221–236. Springer, 2010.
- [To09a] A. W. To. Model checking FO(R) over one-counter processes and beyond. In E. Grädel and R. Kahle, editors, *CSL*, volume 5771 of *Lecture Notes in Computer Science*, pages 485–499. Springer, 2009.
- [To09b] A. W. To. Unary finite automata vs. arithmetic progressions. *Information Processing Letters*, 109(17):1010–1014, 2009.
- [To10] A. W. To. Parikh images of regular languages: Complexity and applications. *CoRR*, abs/1002.1464, 2010.
- [Var95] M. Y. Vardi. An automata-theoretic approach to linear temporal logic. In F. Moller and G. M. Birtwistle, editors, *Banff Higher Order Workshop*, volume 1043 of *Lecture Notes in Computer Science*, pages 238–266. Springer, 1995.
- [Var01] M. Y. Vardi. Branching vs. linear time: Final showdown. In T. Margaria and W. Yi, editors, *TACAS*, volume 2031 of *Lecture Notes in Computer Science*, pages 1–22. Springer, 2001.

- [vD08] D. van Dalen. *Logic and Structures*. Springer, 4th edition, 2008.
- [vLW01] J. H. van Lint and R. M. Wilson. *A Course in Combinatorics*. Cambridge University Press, 2nd edition, 2001.
- [VSS05] K. N. Verma, H. Seidl, and T. Schwentick. On the complexity of equational horn clauses. In R. Nieuwenhuis, editor, *CADE*, volume 3632 of *Lecture Notes in Computer Science*, pages 337–352. Springer, 2005.
- [VW86a] M. Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification (preliminary report). In *LICS*, pages 332–344. IEEE Computer Society, 1986.
- [VW86b] M. Y. Vardi and P. Wolper. Automata-theoretic techniques for modal logics of programs. *Journal of Computer and System Sciences*, 32:183–221, 1986.
- [Wag87] K. W. Wagner. More complicated questions about maxima and minima, and some closures of NP. *Theor. Comput. Sci.*, 51(1-2):53–80, 1987.
- [Wal96] I. Walukiewicz. Pushdown processes: Games and model checking. In R. Alur and T. A. Henzinger, editors, *CAV*, volume 1102 of *Lecture Notes in Computer Science*, pages 62–74. Springer, 1996.
- [Wal00] I. Walukiewicz. Model checking CTL properties of pushdown systems. In S. Kapoor and S. Prasad, editors, *FSTTCS*, volume 1974 of *Lecture Notes in Computer Science*, pages 127–138. Springer, 2000.
- [Wal01] I. Walukiewicz. Pushdown processes: Games and model-checking. *Inf. Comput.*, 164(2):234–263, 2001.
- [Wal02] I. Walukiewicz. Monadic second-order logic on tree-like structures. *Theor. Comput. Sci.*, 275(1-2):311–346, 2002.
- [WB98] P. Wolper and B. Boigelot. Verifying systems with infinite but regular state spaces. In Hu and Vardi [HV98], pages 88–97.
- [WB00] P. Wolper and B. Boigelot. On the construction of automata from linear arithmetic constraints. In Graf and Schwartzbach [GS00], pages 1–19.

- [Wol00] P. Wolper. Constructing automata from temporal logic formulas: A tutorial. In E. Brinksma, H. Hermanns, and J.-P. Katoen, editors, *European Educational Forum: School on Formal Methods and Performance Analysis*, volume 2090 of *Lecture Notes in Computer Science*, pages 261–277. Springer, 2000.
- [WT07] S. Wöhrle and W. Thomas. Model checking synchronized products of infinite transition systems. *Logical Methods in Computer Science*, 3(4), 2007.
- [Yen92] H.-C. Yen. A unified approach for deciding the existence of certain Petri net paths. *Inf. Comput.*, 96(1):119–137, 1992.
- [YKB09] T. Yavuz-Kahveci and T. Bultan. Action Language Verifier: an infinite-state model checker for reactive software specifications. *Formal Methods in System Design*, 35(3):325–367, 2009.
- [YKBB05] T. Yavuz-Kahveci, C. Bartzis, and T. Bultan. Action Language Verifier, extended. In K. Etessami and S. K. Rajamani, editors, *CAV*, volume 3576 of *Lecture Notes in Computer Science*, pages 413–417. Springer, 2005.
- [Zie07] G. M. Ziegler. *Lectures on Polytopes*. Springer, 2007.

Appendix A

Proofs from Chapter 4

A.1 Proposition 4.1.5 implies necessity in Lemma 4.1.4

We now complete the proof of necessity in Lemma 4.1.4 by inductively constructing the desired sequences $\{\alpha_i\}_{i \in \mathbb{N}}$ and $\{\beta_i\}_{i \in \mathbb{N}}$ by using Proposition 4.1.5 at every induction step. In the following, a sequence $\{\eta_i\}_{i \in \mathbb{N}}$ of paths of \mathcal{A} is said to be *good* if $\eta_0(0) = q_0$ and $\mathbf{last}(\eta_i) = \mathbf{first}(\eta_{i+1})$ for all $i \in \mathbb{N}$. In other words, the sequence of paths is good if they can be concatenated to form a run in \mathcal{A} . The same notion can similarly be defined for sequences of paths of \mathcal{R} . So, given a word $v \in \Sigma^*$, suppose that $v \in \text{Rec}_{\rightarrow}(\mathcal{L}(\mathcal{A})) = \text{CHAIN}(\mathcal{L}(\mathcal{A}), \mathcal{L}(\mathcal{R}))$.

Claim. There exist two sequences $\{\alpha_i\}_{i \geq 0}$ and $\{\beta_i\}_{i \geq 0}$ of words, a good sequence $\{\eta_i\}_{i \geq 0}$ of paths of \mathcal{A} , and a good sequence $\{\theta_i\}_{i \geq 0}$ of paths of \mathcal{R} such that $\alpha_0 = v$, $\eta_0 = q_0$, $\theta_0 = q'_0$, and for all $k \in \mathbb{N}$:

1. for all $0 < i \leq k$, $|\alpha_i| > 0$,
2. for all $0 \leq i < k$, $|\beta_i| = |\alpha_i|$,
3. for all $0 \leq i \leq k$, $\pi_i := \eta_0 \odot \dots \odot \eta_i$ is a run of \mathcal{A} on $\beta_0 \dots \beta_{i-1}$,
4. for all $0 \leq i \leq k$, $\pi'_i := \theta_0 \odot \dots \odot \theta_i$ is a run of \mathcal{R} on $(\beta_0 \otimes \beta_0) \dots (\beta_{i-1} \otimes \beta_{i-1})$,
5. for all $0 < i \leq k$, \mathcal{A} accepts α_i from q_i , where $q_i = \mathbf{last}(\pi_i)$,
6. for all $0 \leq i < k$, \mathcal{R} accepts $\alpha_i \otimes \beta_i \alpha_{i+1}$ from q'_i , where $q'_i = \mathbf{last}(\pi'_i)$,
7. for all $0 \leq i \leq k$, $\alpha_i \in \text{CHAIN}(\mathcal{L}(\mathcal{A}^{q_i}), \mathcal{L}(\mathcal{R}^{q'_i}))$, where $q_i = \mathbf{last}(\pi_i)$ and $q'_i = \mathbf{last}(\pi'_i)$.

Observe that this claim immediately implies Lemma 4.1.4 as we may simply define $\pi = \eta_0 \odot \eta_1 \odot \dots$ and $\pi' = \theta_0 \odot \theta_1 \odot \dots$. To prove this claim, we shall define these four sequences inductively. For each $k \in \mathbb{N}$, we shall define four partial sequences $\{\alpha_i\}_{0 \leq i \leq k}$, $\{\beta_i\}_{0 \leq i < k}$, $\{\eta_i\}_{0 \leq i \leq k}$, and $\{\theta_i\}_{0 \leq i \leq k}$ satisfying the conditions in the claim. We shall first deal with the base case $k = 0$. We define $\alpha_0 = v$, $\eta_0 = q_0$, and $\theta_0 = q'_0$. It is easy to see that statements (1),(2),(5), and (6) are vacuous. Statements (3)–(4) are also true because q_0 (resp. q'_0) is a run of \mathcal{A} (resp. \mathcal{R}) on ε . Statement (7) is true by assumption that $v \in \text{CHAIN}(\mathcal{L}(\mathcal{A}), \mathcal{L}(\mathcal{R}))$. Assume now that $k > 0$ and the four partial sequences have been defined satisfying the seven conditions in the claim for all natural numbers up to k . We shall now extend these partial sequences by defining α_{k+1} , β_k , η_{k+1} , and θ_{k+1} . By induction, we have $\alpha_k \in \text{CHAIN}(\mathcal{L}(\mathcal{A}^{q_k}), \mathcal{L}(\mathcal{R}^{q'_k}))$ and so Proposition 4.1.5 gives us a word $v'v''$. We may set $\beta_k = v'$ and $\alpha_{k+1} = v''$. It is immediate that condition (1) and (2) are satisfied. We define η_{k+1} to be the prefix of length $|v'|$ of the run ρ of \mathcal{A}^{q_k} on $v'v''$ given by Proposition 4.1.5. We define θ_{k+1} to be the run ρ' of $\mathcal{R}^{q'_k}$ of length $|v'|$ given by Proposition 4.1.5. It is easy to see now that condition (3)–(5) hold whenever $i = k + 1$ and condition (6) hold whenever $i = k$. Proposition 4.1.5 also implies that $\alpha_{k+1} \in \text{CHAIN}(\mathcal{L}(\mathcal{A}^{q_{k+1}}), \mathcal{L}(\mathcal{R}^{q'_{k+1}}))$, where $q_{k+1} := \mathbf{last}(\rho_{k+1})$ and $q'_{k+1} := \mathbf{last}(\rho'_{k+1})$. Finally, conditions (3)–(7) hold for other smaller values of i by induction. This completes our proof for the claim and therefore the proof of Lemma 4.1.4.

A.2 Proof of Lemma 4.2.2

Suppose that $T_1 = (D_1, \tau_1)$ and $T_2 = (D_2, \tau_2)$. Let us first prove existence. The context tree $T = (D, \tau)$ is defined as follows. Let

$$D = (D_1 \cap D_2) \cup \{vi \in D_1 \setminus D_2 : i \in \Upsilon, v \in D_2\}.$$

In other words, the tree domain D contains all nodes that are both in D_1 and D_2 and additionally the children of the nodes $v \in D_1 \cap D_2$ with respect to the tree T_1 but which do not belong to D_2 . The node labeling τ is defined as follows: for each $v \in D_1 \cap D_2$, $\tau(v) := \tau_1(v)$; for other nodes $u_1, \dots, u_n \in D_1 \setminus D_2$, we assign $\tau(u_i) := x_i$. It is clear that both conditions are satisfied.

To prove uniqueness, consider another context tree $T' = (D', \tau')$ satisfying the two prescribed conditions. We shall now prove that $T = T'$ up to relabeling of the context leaves. We first show that $D = D'$. To show $D' \subseteq D$, observe that condition (2) implies

that $D' \subseteq D_1$. Let $v \in D'$. If $v \in D_2$, then we are done; otherwise, condition (1) implies that v must be a context leaf in T' of the form ui for some $u \in D_2$ and $i \in \Upsilon$. In any case, we have $v \in D$. Conversely, we also have $D \subseteq D'$. To see this, observe that $D_1 \cap D_2 \subseteq D'$; for, otherwise, if $u \in D_1 \cap D_2$ such that $u \notin D'$ and v is the longest prefix of u satisfying $v \in D_1 \cap D_2 \cap D'$ (which must exist since $\varepsilon \in D'$), both conditions imply that v is a context leaf but then condition (1) implies that $v \notin D_2$, which results in a contradiction. Furthermore, each node $vi \in D_1 \setminus D_2$ such that $i \in \Upsilon$ and $v \in D_2$ must be in D' as a context leaf in T' by condition (2). Finally, we note that this proof also implies that the context leaves of T are precisely the context leaves of T' . Therefore, τ and τ' coincide except when evaluated on the context leaves. This completes the proof of uniqueness.

A.3 Proof of Proposition 4.3.4

The proof is via a reduction from the nonemptiness problem for language intersections of DWAs, which is PSPACE-complete [GJ79]. More precisely, the problem is to decide whether, given DWAs $\mathcal{A}_1, \dots, \mathcal{A}_n$ over some alphabet Σ , the language $\mathcal{L}(\mathcal{A}_1) \cap \dots \cap \mathcal{L}(\mathcal{A}_n)$ is nonempty.

The proof is rather simple. From the input automata $\mathcal{A}_1, \dots, \mathcal{A}_n$, we construct a PDS which simply guesses a word $w \in \Sigma^*$ which witnesses the intersection of the languages $\mathcal{L}(\mathcal{A}_1), \dots, \mathcal{L}(\mathcal{A}_n)$. More precisely, let $\mathcal{P} = (\text{ACT}, \Gamma, Q, \delta)$ be the PDS defined as:

- $\text{ACT} = \{\alpha\}$,
- $\Gamma = \Sigma \cup \{\perp\}$,
- $Q = \{q_1, q_2\}$,
- $\delta = \{((q_1, \alpha, a), (q_1, ab)) : a \in \Gamma, b \in \Sigma\} \cup \{(q, \alpha, a), (q_2, a)) : q \in Q, a \in \Gamma\}$.

For each \mathcal{A}_i , let \mathcal{B}_i be the DWA that recognizes the language $\{q_2 \perp w : w \in \mathcal{L}(\mathcal{A}_i)\}$. It is easy to see that each \mathcal{B}_i can be constructed in polynomial time. Observe now that $\bigcap_{i=1}^n \mathcal{L}(\mathcal{A}_i) \neq \emptyset$ iff $q_1 \perp \in \text{Rec}(\mathcal{L}(\mathcal{B}_1), \dots, \mathcal{L}(\mathcal{B}_n))$. This is because *every* infinite run from $q_1 \perp$ in the transition system $\mathfrak{S}_{\mathcal{P}}$ generated by \mathcal{P} that visits $\mathcal{L}(\mathcal{B}_1), \dots, \mathcal{L}(\mathcal{B}_n)$ infinitely often must eventually self-loop at some configuration of the form $q_2 \perp w$ for some word $w \in \Sigma^*$. Conversely, the existence of a witness word $w \in \bigcap_{i=1}^n \mathcal{L}(\mathcal{A}_i)$ implies the existence of a run in $\mathfrak{S}_{\mathcal{P}}$ from $q_1 \perp$ which visits $q_2 \perp w$, which then self-loops forever.

A.4 Proof of Proposition 4.3.7

The proof is via a reduction from the nonemptiness problem for language intersections of NTAs, which is EXP-complete [CDG⁺07]. More precisely, the problem is to decide whether, given NTAs $\mathcal{A}_1, \dots, \mathcal{A}_n$ over $\text{TREE}_2(\Sigma)$, the language $\mathcal{L}(\mathcal{A}_1) \cap \dots \cap \mathcal{L}(\mathcal{A}_n)$ is nonempty.

The polynomial-time reduction is very similar to the proof of Proposition 4.3.4, and hence we shall only sketch it. We shall construct a GTRS \mathcal{P} over $\text{TREE}_2(\Sigma')$, where Σ' is the union of Σ with a set containing a special “guessing” node label q_1 , and a special “sink” node label q_2 . The GTRS \mathcal{P} will start at a tree with only a single node labeled q_1 , and “guesses” a tree T that witnesses nonemptiness of the language intersection problem. At any given point, each leaf will either be labeled q_1 or q_2 . We will also have rules of the form $q_1 \rightarrow q_2$ and $q_2 \rightarrow q_2$. The input NTAs $\mathcal{A}_1, \dots, \mathcal{A}_n$ will be slightly modified by attaching leaf nodes labeled q_2 to the leaves of each tree recognized by these automata. The rest is identical to the proof of Proposition 4.3.4.

Appendix B

Proofs from Chapter 5

B.1 Proof of Proposition 5.4.6

We show that model checking the negations of LTL_{det} formulas over GTRS is NP-hard. Our reduction is from the problem HAMPATH of testing whether there exists a hamiltonian path in a directed graph. Suppose the input is a graph $G = (V, E)$ with vertices $V = \{1, \dots, n\}$ and edges $E \subseteq V \times V$. Node labels for our trees in the output GTRS \mathcal{P} will draw from the set $\Sigma := \{X, \text{root}, \text{eval}\} \cup \{1, \dots, n\}$. The initial tree t_0 to be evaluated against the input LTL_{det} formula is the tree drawn in Figure B.1. We have action labels $\text{ACT} := \{1, \dots, n\} \cup \{\text{eval}, \text{fin}\}$. We now describe the transition rule of \mathcal{P} . For each $i \in \{1, \dots, n\}$, we have a rule $X \rightarrow_i i$. For each directed edge $(u, v) \in E$, we have a transition rule $\text{eval}(u, v) \rightarrow_{\text{eval}} u$, where $\text{eval}(u, v)$ is a notation for the tree with three nodes with root labeled eval , left child labeled u , and right child labeled v . For each $i \in \{1, \dots, n\}$, we also have a transition rule $\text{root}(i) \rightarrow_{\text{fin}} \text{root}(i)$, where $\text{root}(i)$ is the tree with two nodes with root labeled root and only child labeled i . Let ϕ_{eval} be the formula

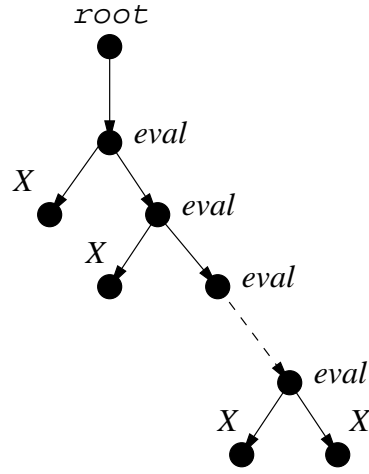
$$\text{eval} \wedge \mathbf{X}(\text{eval} \wedge \mathbf{X}(\text{eval} \dots \wedge \mathbf{X}(\text{eval} \wedge \text{fin})))$$

where the number of eval $n - 1$. Let ϕ be the formula

$$1 \wedge \mathbf{X}(2 \dots \wedge \mathbf{X}(n \wedge \mathbf{X}\phi_{\text{eval}})).$$

Claim B.1.1 $(\Lambda(\mathcal{P}), t_0) \not\models \neg\phi$ iff G has a hamiltonian path.

It is not hard to see that the claim is true. A hamiltonian path $\pi := i_1, i_2, \dots, i_n$ in G will correspond to the tree t_π where the j th leaf is “evaluated” as i_j , and vice versa.

Figure B.1: The initial tree t_0 with n leaf nodes.

Finally, it is not hard to see that the negation of φ is an LTL_{det} formula (and can be computed in poly-time). To show this, it suffices to show that the negation of $\psi := p \wedge \mathbf{X}(\varphi)$ is an LTL_{det} formula, provided that $\neg\varphi$ is an LTL_{det} formula. In fact, if $p \in \text{ACT}$, the formula ψ is equivalent to $p \wedge (\neg p \vee \mathbf{X}(\varphi))$, and hence $\neg\psi$ is equivalent to $\neg p \vee (p \wedge \mathbf{X}(\neg\varphi))$ which is an LTL_{det} (since $\neg\varphi$ is an LTL_{det} formula).

Appendix C

Proofs from Chapter 7

C.1 Proof of Fact 7.3.3

The proof is by induction on the length $|\pi|$ of the path π . Whenever $|\pi| = 0$ (it contains only a single state), we may then take $\pi' = \pi$ and set $h = 0$. Then, we have $\mathcal{P}(\pi) = \mathcal{P}(\pi') + \sum_{i=1}^h \mathcal{P}(C_i)$. Suppose that Fact 7.3.3 holds for all paths up to length $k - 1 \geq 0$. We shall now show that it also holds for all paths of length k . Let π be a path of length k . If π is simple, then we may set $\pi' := \pi$ and $h := 0$, and the proof is complete. Therefore, assume that π is not simple and let $\pi = p_0 \dots p_k$, where $p_0 = q$ and $p_k = q'$. Let $i \geq 0$ be any index such that the state p_i appears in π more than once, say, at p_i and p_j . Then, consider the path

$$\pi_1 := p_0 p_1 \dots p_{i-1} p_i p_{j+1} p_{j+2} \dots p_k$$

of length strictly smaller than k that is obtained by removing the segment $\pi[i + 1, j]$ from π . We will now apply the induction hypothesis twice. Firstly, applying the induction hypothesis on the path π_1 , we obtain a simple path π' and finitely many simple cycles C_1, \dots, C_h possibly with duplicates such that

$$\mathcal{P}(\pi_1) = \mathcal{P}(\pi') + \sum_{i=1}^h \mathcal{P}(C_i).$$

We now apply the induction hypothesis again on the path $\pi[i + 1, j]$ from p_i to p_j that is of length smaller than k yielding a simple path p_i of length 0 (since $p_i = p_j$) and finitely many simple cycles C'_1, \dots, C'_r such that

$$\mathcal{P}(\pi[i + 1, j]) = \mathcal{P}(p_i) + \sum_{i=1}^r \mathcal{P}(C'_i) = \sum_{i=1}^r \mathcal{P}(C'_i).$$

This clearly implies that

$$\mathcal{P}(\pi) = \mathcal{P}(\pi') + \sum_{i=1}^h \mathcal{P}(C_i) + \sum_{j=1}^r \mathcal{P}(C'_j),$$

which extends the validity of Fact 7.3.3 to value k . Therefore, by mathematical induction, Fact 7.3.3 holds for all values of k , which completes the proof.

C.2 Proof of Lemma 7.3.5

We only show that if the $M_{\mathbf{v}}[j, h] = 1$, then so is the (j, h) -component of the matrix on the r.h.s. The converse can be proved by observing that all the steps below can be easily reversed.

Let $s = 1 + \sum_{i=1}^k r_i$. Suppose that $\pi = q_{l_0} b_1 q_{l_1} \dots b_s q_{l_s}$ is a path from q_j to q_h with $\mathcal{P}(\pi) = \mathbf{v}$. Thus, we have $l_0 = j$ and $l_s = h$. We now decompose π as follows. Let t be the first position where the letter a_i occurs in π , i.e., $b_t = a_i$ and $b_{t'} \neq a_i$ for all $t' < t$. Let $\pi_1 := q_{l_0} b_1 \dots q_{l_{t-1}}$, $\pi_2 := q_{l_{t-1}} b_t q_{l_t}$, and $\pi_3 := q_{l_t} b_{t+1} \dots q_{l_s}$. Let $\mathbf{u} := \mathcal{P}(\pi_1)$ and $\mathbf{w} := \mathcal{P}(\pi_3)$. Notice that the i th entry of \mathbf{u} is 0 and $\mathbf{w} = \mathbf{v} - \mathbf{e}_i - \mathbf{u}$. Furthermore, we have $M_{\mathbf{u}}[j, l_{t-1}] = M_{\mathbf{e}_i}[l_{t-1}, l_t] = M_{\mathbf{w}}[l_t, h] = 1$. It follows that the (j, h) -component of the matrix $M_{\mathbf{u}} \bullet M_{\mathbf{e}_i} \bullet M_{\mathbf{w}}$ is 1.

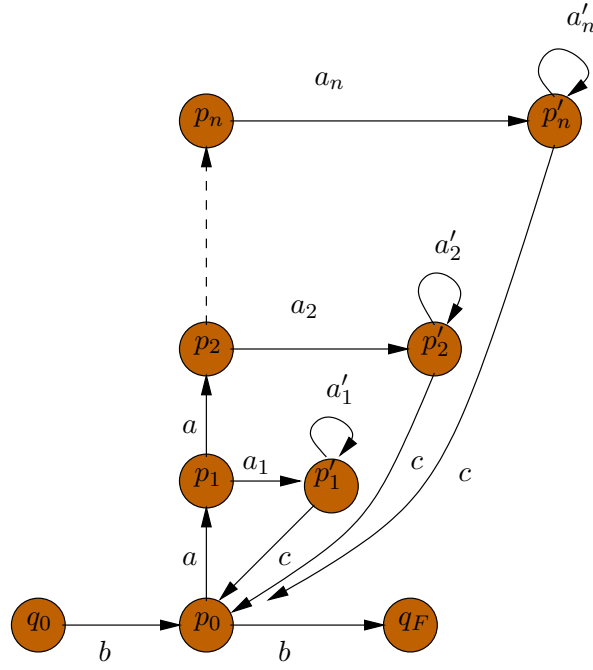
C.3 Proof of Lemma 7.3.6

This can be done using Lemma 7.3.5 and dynamic programming. The algorithm has $n + 1$ stages. At stage $j = 0, \dots, n$, we compute all $M_{\mathbf{v}}$ where the components in \mathbf{v} sum up to j . These will be saved in the memory for subsequent stages of the iteration. As base cases, we would obtain $M_{\mathbf{0}}$ and $M_{\mathbf{e}_i}$, for each $1 \leq i \leq k$, directly from the input. Notice that boolean matrix multiplication can be done in $O(n^3)$ and so each stage of the computation can be performed in $O(n^{2k+4})$. Thus, the entire computation runs in time $2^{O(k \log n)}$.

C.4 Proof of Proposition 7.3.10

The automaton $\mathcal{A}_n = (\Sigma_n, Q_n, \delta_n, q_0, q_F)$, where

$$\begin{aligned} Q_n &= \{q_0, q_F\} \cup \{p_0, \dots, p_n\} \cup \{p'_1, \dots, p'_n\}, \\ \Sigma_n &= \{a, b, c\} \cup \{a_i, a'_i : 1 \leq i \leq n\}. \end{aligned}$$

Figure C.1: A depiction of the DWA \mathcal{A}_n

We specify the transition function δ_n in Figure C.1. Notice that \mathcal{A}_n has an equivalent regular expression e_n of size $O(n)$. For example, when $n = 2$, we can define e_n to be

$$b(a(a_1(a'_1)^*c|aa_2(a'_2)^*c))^*b.$$

We now argue that the a -component of some \mathbf{v}_i must be at least $n(n+1)/2$. Let m be the maximum entry over all vectors in $\bigcup_{i=1}^r S_i$. Define

$$N := \left(\max\{|S_i| : 1 \leq i \leq r\} \frac{n(n+1)}{2} m \right) + 1.$$

For each $1 \leq i \leq n$, let C_i be the cycle $p_0 a p_1 a \dots p_i a_i p'_i (a'_i p'_i)^N c p_0$. Consider the accepting path $\pi = (q_0 b p_0) \odot C_1 \odot C_2 \odot \dots \odot C_n \odot (p_0 b q_F)$. We have $\mathcal{P}(\pi) \in P(\mathbf{v}_h; S_h)$ for some $1 \leq h \leq r$. Observe also that a occurs precisely $\sum_{i=1}^n i = n(n+1)/2$ times in π .

Claim C.4.1 *Each a'_i -component of \mathbf{v}_h ($1 \leq i \leq n$) is nonzero.*

We now prove this claim. Let $S_h = \{\mathbf{u}_1, \dots, \mathbf{u}_s\}$ and $\mathcal{P}(\pi) = \mathbf{v}_h + \sum_{i=1}^s t_i \mathbf{u}_i$. For each $i \in \{1, \dots, n\}$, there exists a vector \mathbf{u}_{j_i} with a positive a'_i -component and $t_{j_i} > n(n+1)/2$; for, otherwise, the a'_i -component of $\mathcal{P}(\pi)$ is at most $|S_h| \frac{n(n+1)}{2} m < N$, a contradiction. In particular, this implies that all α -component of \mathbf{u}_{j_i} , where $\alpha \neq a'_i$ for

all $i \in \{1, \dots, n\}$, is 0 as each such letter α occurs at most $n(n+1)/2$ times in π . But this means that each a_i -component of \mathbf{v}_h is nonzero; for, otherwise, we could consider the vector $\mathbf{v}_h + \mathbf{u}_{j_i}$ which would not correspond to any accepting path in \mathcal{A}_n since at least one letter a_i needs to read by \mathcal{A}_n if a'_i is to occur in the path. This proves our claim.

Now consider each word $w = w_0 \dots w_l \in \mathcal{L}(\mathcal{A}_n)$ such that $\mathcal{P}(w) = \mathbf{v}_h$. It is easy to see that the number of occurrences of a in w must be at least $n(n+1)/2$. In fact, for every $1 \leq i \leq n$, define $j_i = \min\{j : w_j = a_i\}$. For each i , the number of occurrences of a in $w_t \dots w_{j_i}$, where $t = \max\{j_{i'} : j_{i'} < j_i, 1 \leq i' \leq n\}$, is at least i . The lower bound of $n(n+1)/2$ on the number of occurrences of a in w immediately follows.

C.5 Proof of Proposition 7.4.2

DWA

We now give a poly-time reduction from the hamiltonian path problem to membership problem for Parikh images of DWAs. The hamiltonian path problem asks whether a given graph $\mathfrak{G} = \langle V = \{v_1, \dots, v_n\}, E \rangle$ has a hamiltonian path from v_1 to v_n , i.e., a path from v_1 to v_n in \mathfrak{G} that visits each vertex in V *exactly* once. Given \mathfrak{G} , we define the DWA $A_{\mathfrak{G}} = (\Sigma, Q, \delta, q_0, q_F)$ where $Q := V$, $\Sigma := \{a_1, \dots, a_n\}$, $q_0 := v_1$, $q_F := v_n$, and $\delta := \{(v_i, a_j, v_j) : (v_i, v_j) \in E\}$. Then, it is easy to see that \mathfrak{G} has a hamiltonian path from v_1 to v_n iff the Parikh image $\mathcal{P}(a_1 \dots a_n)$ of the word $a_1 \dots a_n$ is in $\mathcal{P}(A_{\mathfrak{G}})$ iff $(0, 1, 1, \dots, 1) \in \mathcal{P}(A_{\mathfrak{G}})$. This completes the proof of NP-hardness of the membership problem for Parikh images of DWAs with unbounded alphabet size.

Regular expressions

One-in-three 3SAT is the following problem: given a boolean formula ϕ in 3-CNF, does there exist a satisfying assignment for ϕ that additionally makes no more than one literal true for each clause. We shall call such a satisfying assignment *1-in-3*. This problem is NP-complete (cf. see [GJ79]). We shall reduce this problem to the membership problem for Parikh images of regular expressions. Given $\phi = C_1 \wedge \dots \wedge C_k$, where C_i is a multiset over $L := \{x_1, \neg x_1, \dots, x_n, \neg x_n\}$ with $|C_i| = 3$, we define a function $f : L \rightarrow \{1, \dots, k\}^*$ as follows:

- $f(x_i) := a_1 \dots a_k$ where $a_i := i$ if $x_i \in C_j$, and $a_i := \varepsilon$ otherwise.

- $f(\neg x_i) := a_1 \dots a_k$ where $a_i := i$ if $\neg x_i \in C_j$, and $a_i := \varepsilon$ otherwise.

That is, the function f associates a literal with the indices of clauses that are satisfied when the value 1 is assigned to the literal. The corresponding regular expression over $\Sigma = \{1, \dots, k\}$ is

$$e_\varphi = (f(x_1)|f(\neg x_1)) \dots (f(x_n)|f(\neg x_n)).$$

Let $\mathbf{1} \in \{1\}^k$. We claim that φ is a positive instance of one-in-three 3SAT iff $\mathbf{1} \in \mathcal{P}(\mathcal{L}(e_\varphi))$. To prove this, suppose that φ is a positive instance with a 1-in-3 satisfying assignment $\sigma : L \rightarrow \{0, 1\}$ (i.e. $\sigma(x_i) = 1$ iff $\sigma(\neg x_i) = 0$). Consider the word $w := X_1 \dots X_n \in \Sigma^*$, where

$$X_i := \begin{cases} f(x_i) & \text{if } \sigma(x_i) = 1, \\ f(\neg x_i) & \text{if } \sigma(x_i) = 0. \end{cases}$$

Observe that $w \in \mathcal{L}(e_\varphi)$. Since σ is a 1-in-3 satisfying assignment, it follows that $\mathcal{P}(w) = \mathbf{1}$ and, therefore, we have $\mathbf{1} \in \mathcal{P}(\mathcal{L}(e_\varphi))$. The converse direction can be proved by reversing the above construction of the word w . Finally, observe that the construction of e_φ and $\mathbf{1}$ can be done in time polynomial in the size of φ .

C.6 Proof of Proposition 7.5.7

This result is almost immediate from the undecidability of the emptiness problem for deterministic 0-reversal 3-counter systems that *may test equality of the current values of two counters* [ISD⁺02]. More precisely, such a counter system is of the form $\mathcal{M} = (\text{ACT}, X, Q, \Delta)$ with $X = \{x, y, z\}$ and instructions $((q, \varphi(X)), a, (q', i_1, \dots, i_k))$ of the form:

- for each $j \geq 1$, $i_j \geq 0$, and
- $\varphi(X)$ is the (guard) Presburger formula $z \sim x$ or $z \sim y$, where \sim is either $=$ or \neq .

The semantics is similar to the usual notion of counter systems. For example, if the instruction has a guard formula $z = x$, the machine simply tests whether the current values held by these variables coincide and then the instructions can be executed if the test was successful. The *emptiness problem* corresponds to the problem of deciding whether the configuration $(q_0, 0, 0, 0)$ of \mathcal{M} , where $q_0 \in Q$ is a designated initial state of \mathcal{M} , may reach a designated final state $q_F \in Q$.

We now give the reduction. On an input counter system \mathcal{M} of the above form, our new counter system $\mathcal{M}' = (\text{ACT}', X, Q', \Delta')$ will be 1-reversal and not have comparison

tests between the values of two different counters. Intuitively, when simulating an instruction in \mathcal{M} of the form $((q, \alpha \sim \beta), a, (q', i_1, i_2, i_3))$, the new counter system \mathcal{M}' will move from the state q to the intermediate state $(q, \alpha \sim \beta)$ and then move to the state q' while modifying the counters using i_1, i_2, i_3 and *ignoring the test* $\alpha \sim \beta$. Our CTL formula will later take care of the test $\alpha \sim \beta$ using one extra reversal. More precisely, the new counter system $\mathcal{M}' = (\text{ACT}', X, Q', \Delta')$ is defined as follows:

- $\text{ACT}' = \{\star, \ominus, \text{success}\}$,
- $X = \{x, y, z\}$,
- Q' is a union of Q , $\{(q, \varphi(X)) : q \in Q, \text{ and } \varphi \text{ is a guard formula in } \mathcal{M}\}$, and $\{(test, \varphi(X)) : \varphi \text{ is a guard formula in } \mathcal{M}\}$, where *test* is a new state (i.e. does not occur in Q), and
- Δ' contains instructions of the form:
 - $((q_F, \top), \text{success}, (q_F, 0, 0, 0))$.
 - $((q, \top), \ominus, ((q, \varphi), 0, 0, 0))$, if there is an instruction in \mathcal{M} of the form

$$((q, \varphi), a, (q', i_1, i_2, i_3)).$$
 - $((q, \varphi), \top, \ominus, (q', i_1, i_2, i_3))$, if there is an instruction in \mathcal{M} of the form $((q, \varphi), a, (q', i_1, i_2, i_3))$.
 - $((q, \varphi), \top, \ominus, ((test, \varphi), 0, 0, 0))$.
 - $(test, \alpha \sim \beta)$ may loop on the same state with the action \ominus while decrementing α and β by 1.
 - $(test, \alpha = \beta)$ may test whether $\alpha = 0$ and $\beta = 0$ and loop on the same state with the action \star without modifying the counters.
 - $(test, \alpha \sim \beta)$ may test whether $\alpha = 0$ and $\beta > 0$ (or $\alpha > 0$ and $\beta = 0$) and loop on the same state with the action \star without modifying the counters.

The desired CTL formula is

$$\theta = E(\langle \ominus \rangle (\bigvee_{\varphi} (test, \varphi) \rightarrow EF\star) \cup success),$$

where φ is either $x \sim z$ or $y \sim z$, where \sim is either $=$ or \neq . The formula makes sure that each guess made by the counter system regarding the comparison tests between values of counters is correct. It is easy to check that $\mathfrak{S}_{\mathcal{M}'}, (q, 0, 0, 0) \models \theta$ iff $\langle \mathcal{M}, q_0, q_F \rangle$ is a positive instance of the original problem. This completes the reduction.

Appendix D

Proofs from Chapter 8

D.1 Proof of Lemma 8.2.2

Let $\alpha = (\alpha_i)_{i \in [l]}$ be an extended MMA formula. Define the set of (possibly) *invoked pairs* $P \subseteq [l] \times \mathbb{Z}$ to be the set $\{(i, j) : i \in [l], j \in [-l + i, l - i]\} \subseteq [l] \times [-l, l]$. Now associate the set P with the usual lexicographic ordering $\prec_{\mathbb{Z} \times \mathbb{Z}}$ for $\mathbb{Z} \times \mathbb{Z}$ (i.e. $(x, y) \prec_{\mathbb{Z} \times \mathbb{Z}} (x', y')$ iff (1) $x < x'$, or $x = x'$ and $y < y'$). We will now construct a MMA formula

$$\beta = (\beta_{(i,d)})_{(i,d) \in P}$$

such that

$$\llbracket \beta_{(i,d)} \rrbracket = \{x \in \mathbb{N} \mid x + d \geq 0 \wedge x + d \in \llbracket \alpha_i \rrbracket\} \quad (*)$$

for each $(i, d) \in P$.

Recall that we identify a dag $([l], \prec_\alpha)$ with α and its corresponding strict partial order \prec_α^+ . In the following, we will give the definition of $\beta_{(i,d)}$ and directly prove that equation $(*)$ holds, for each $(i, d) \in P$ by induction on i with respect to \prec_α^+ . The lemma will follow from the latter, since we will have $\llbracket \alpha \rrbracket = \llbracket \alpha_l \rrbracket \stackrel{(*)}{=} \llbracket \beta_{(l,0)} \rrbracket = \llbracket \beta \rrbracket$.

For convenience, we will allow formulas of the form $\sim -k$ for positive integer k . Such formulas, which are *strictly speaking* not MMA formulas by definition, serve only as abbreviations. In particular, we can replace abbreviation $\leq -k$ by $\neg(\geq 0)$ (not satisfied by all $k \in \mathbb{N}$) and $\geq -k$ by the formula ≥ 0 (satisfied by all $k \in \mathbb{N}$). For the induction base, assume i is minimal with respect to \prec_α^+ . Take any $d \in [-l, l]$ satisfying $(i, d) \in P$.

- In case $\alpha_i = (\equiv m \bmod n)$ for some $m, n \in \mathbb{N}$. Let $v = (m - d) \bmod n$. Then,

we put

$$\beta_{(i,d)} = (\geq -d) \wedge (\equiv v \pmod n).$$

It is obvious to see that equation $(*)$ holds for (i,d) .

- In case $\alpha_i = (\sim n)$, where $\sim \in \{\leq, \geq\}$ and where $n \in \mathbb{N}$. Define the integer $v = n - d$. Then we put

$$\beta_{(i,d)} = (\geq -d) \wedge (\sim v).$$

Again, it is obvious to see that equation $(*)$ holds (i,d) .

For the induction step, assume i is not minimal with respect to \prec_α^+ . Take any number $d \in [-l, l]$ satisfying $(i,d) \in P$. Again, we make a case distinction according to α_i .

- Assume $\alpha_i = \neg\alpha_j$ for some $j \in [i-1]$. Then, we put

$$\beta_{(i,d)} = (\geq -d) \wedge (\neg\beta_{(j,d)}).$$

Observe that this is possible, since $(j,d) \in P$. By induction hypothesis, we have

$$\llbracket \beta_{(j,d)} \rrbracket = \{x \in \mathbb{N} \mid x+d \geq 0 \wedge x+d \in \llbracket \alpha_j \rrbracket\}.$$

Hence,

$$\llbracket \neg\beta_{(j,d)} \rrbracket = \{x \in \mathbb{N} \mid x+d < 0 \vee x+d \notin \llbracket \alpha_j \rrbracket\}$$

which equals

$$\{x \in \mathbb{N} \mid x+d < 0 \vee x+d \in \llbracket \alpha_i \rrbracket\}.$$

The latter and the definition of $\beta_{(i,d)}$ yields that $\llbracket \beta_{(i,d)} \rrbracket$ equals

$$\{x \in \mathbb{N} \mid x+d \geq 0 \wedge (x+d < 0 \vee x+d \in \llbracket \alpha_i \rrbracket)\}.$$

which equals

$$\{x \in \mathbb{N} \mid x+d \geq 0 \wedge x+d \in \llbracket \alpha_i \rrbracket\}.$$

delivering $(*)$ for (i,d) as required.

- Assume $\alpha_i = \alpha_j \wedge \alpha_k$ for some $j, k \in [i-1]$. Then we put

$$\beta_{(i,d)} = \beta_{(j,d)} \wedge \beta_{(k,d)}.$$

Again, this is possible, since $(j,d), (k,d) \in P$. That equation $(*)$ holds for (i,d) follows directly from induction hypothesis.

- Assume $\alpha_i = n \sim \min \alpha_j$ for some $\sim \in \{\leq, \geq\}$, $j \in [i-1]$, and $n \in \mathbb{N}$. Then, we put

$$\beta_{(i,d)} = (\geq -d) \wedge n \sim \min \beta_{(j,0)}.$$

Then, (*) is immediate by induction hypothesis.

- Assume $\alpha_i = n \sim \max(\alpha_j, n')$ for some $\sim \in \{\leq, \geq\}$, $j \in [i-1]$, and $n, n' \in \mathbb{N}$. Then, we put

$$\beta_{(i,d)} = (\geq -d) \wedge n \sim \max(\beta_{(j,0)}, n').$$

Then, (*) is immediate by induction hypothesis.

- Assume $\alpha_i = \sim \min \alpha_j$ for some $\sim \in \{\leq, \geq\}$ and for some $j \in [i-1]$. Then we put $\beta_{(i,d)}$ as

$$(\geq -d \wedge d > \min \beta_{(j,0)} \wedge \sim -1) \vee$$

$$(\geq -d \wedge d \leq \min \beta_{(j,0)} \wedge \sim \min \beta_{(j,d)}).$$

To see why this definition is correct, we first inspect the right hand side of the equation (*). Observe that for all $k \in \mathbb{N}$, we have $k + d \sim \llbracket \alpha_j \rrbracket$ and $k + d \geq 0$ iff either of the following two statements hold: (1) $k + d \geq 0$, $d > \llbracket \alpha_j \rrbracket$ and $k \sim -1$, or (2) $k + d \geq 0$, $d \leq \llbracket \alpha_j \rrbracket$, and $k \sim \min \llbracket \alpha_j \rrbracket - d$. This is clearly because for all $k \in \mathbb{N}$ and a negative integer k' we have $k \sim -1$ iff $k \sim k'$. Our definition of $\beta_{(i,d)}$ now implies that $\llbracket \beta_{(i,d)} \rrbracket$ is a union of the set

$$\{k \in \mathbb{N} : k + d \geq 0 \wedge d > \min \llbracket \beta_{(j,0)} \rrbracket \wedge k \sim -1\}$$

and the set of all $k \in \mathbb{N}$ such that

$$k + d \geq 0 \wedge d \leq \min \llbracket \beta_{(j,0)} \rrbracket \wedge k \sim \min \llbracket \beta_{(j,d)} \rrbracket.$$

By induction, we may assume that

$$\llbracket \beta_{(j,d)} \rrbracket = \{k \in \mathbb{N} : k + d \geq 0 \wedge k + d \in \llbracket \alpha_j \rrbracket\}.$$

Similarly, by induction, we have $\llbracket \beta_{(j,0)} \rrbracket = \llbracket \alpha_j \rrbracket$. Furthermore, we have

$$\begin{aligned} \min \llbracket \alpha_j \rrbracket - d &= \min \{k - d \in \mathbb{Z} : k \in \llbracket \alpha_j \rrbracket\} \\ &= \min \{k \in \mathbb{Z} : k + d \in \llbracket \alpha_j \rrbracket\}. \end{aligned}$$

These imply that $\min \llbracket \alpha_j \rrbracket - d = \llbracket \beta_{(j,d)} \rrbracket$ if $\min \llbracket \alpha_j \rrbracket \geq d$. That (*) holds is then immediate.

- $\alpha_i = \sim \max(\alpha_j, c)$ for some $\sim \in \{\leq, \geq\}$, $j \in [i-1]$, and a constant $c \in \mathbb{N}$. Then, letting $v = c - d$, we put $\beta_{(i,d)}$ as

$$\begin{aligned} & (\geq -d \wedge d > \max(\beta_{(j,0)}, c) \wedge \sim -1) \vee \\ & (\geq -d \wedge d \leq \max(\beta_{(j,0)}, c) \wedge \sim \max(\beta_{(j,d)}, v). \end{aligned}$$

The proof is identical to the previous case.

- Assume $\alpha_i = \alpha_j - 1$ for some $j \in [i-1]$. Then we put

$$\beta_{(i,d)} = \beta_{(j,d+1)}.$$

Notice that $(j, d+1) \in P$ since $j < i$ and $(i, d) \in P$. By induction hypothesis, $\llbracket \beta_{(j,d+1)} \rrbracket$ equals

$$\{x \in \mathbb{N} \mid x + d + 1 \geq 0 \wedge x + d + 1 \in \llbracket \alpha_j \rrbracket\}.$$

which is equivalent to

$$\{x \in \mathbb{N} \mid x + d + 1 \geq 0 \wedge x + d \in \llbracket \alpha_j - 1 \rrbracket\}.$$

It follows now that $\llbracket \beta_{(i,d)} \rrbracket$ equals

$$\{x \in \mathbb{N} \mid x + d \geq 0 \wedge x + d \in \llbracket \alpha_i \rrbracket\}.$$

and thus $(*)$ follows for (i, d) .

- Assume $\alpha_i = \alpha_j + 1$ for some $j \in [i-1]$. We put

$$\beta_{(i,d)} = (\geq -d) \wedge \beta_{(j,d-1)}.$$

Notice that $(j, d-1) \in P$ since $j < i$ and $(i, d) \in P$. By induction hypothesis we have that $\llbracket \beta_{(j,d-1)} \rrbracket$ is

$$\{x \in \mathbb{N} \mid x + d - 1 \geq 0 \wedge x + d - 1 \in \llbracket \alpha_j \rrbracket\}.$$

By definition of $\beta_{(i,d)}$ we have that

$$\llbracket \beta_{(i,d)} \rrbracket = \{x \in \mathbb{N} \mid x + d \geq 0 \wedge x \in \llbracket \beta_{(j,d-1)} \rrbracket\}$$

which hence equals

$$\{x \in \mathbb{N} \mid x + d \geq 0 \wedge x + d \geq 1 \wedge x + d - 1 \in \llbracket \alpha_j \rrbracket\}$$

which equals

$$\{x \in \mathbb{N} \mid x + d \geq 0 \wedge x + d \in \llbracket \alpha_i + 1 \rrbracket\}$$

delivering $(*)$ for (i, d) as required. \square

D.2 Proof of Lemma 8.3.1

(1) \Rightarrow (2): Follows trivially since \mathcal{P}' is obtained from \mathcal{P} by adding λ -transitions.

(2) \Rightarrow (3): Let $\pi = s_1 \rightarrow_{\mathcal{P}'} s_2 \cdots \rightarrow_{\mathcal{P}'} s_k$ be a path in $\mathfrak{S}'_{\mathcal{P}}$. Formally, we call π *normalized* if

$$s_1 \downarrow_{\mathcal{P}'} \cdots \downarrow_{\mathcal{P}'} s_i \uparrow_{\mathcal{P}'} \cdots \uparrow_{\mathcal{P}'} s_k$$

for some $i \in [k]$. We claim that every shortest path from s to t is normalized. To prove the implication, assume, by contradiction, that there is a shortest path $\pi = (q_1, n_1) \rightarrow_{\mathcal{P}'} (q_2, n_2) \cdots \rightarrow_{\mathcal{P}'} (q_k, n_k)$ with $s = (q_1, n_1)$ and $t = (q_k, n_k)$ that is not normalized. Then there exists a subpath p

$$(q_i, n_i) \uparrow_{\mathcal{P}'} (q_{i+1}, n_{i+1}) \uparrow_{\mathcal{P}'} \cdots \uparrow_{\mathcal{P}'} (q_j, n_j) \downarrow_{\mathcal{P}'} (q_{j+1}, n_{j+1})$$

in π such that $n_{i+1} = n_{i+2} = \cdots = n_j$ and $n_i = n_{j+1} = n_j + 1$. If, on the one hand $i + 1 = j$, then by rule (R2), it follows that $(q_i, n_i) \rightarrow_{\mathcal{P}'} (q_{j+1}, n_{j+1})$ by which p can be replaced, contradicting the minimality of $|\pi|$. On the other hand, if $i + 1 < j$, then by successively applying rules (R1) and (R4), we obtain that $(q_{i+1}, n_{i+1}) \rightarrow_{\mathcal{P}'} (q_j, n_j)$. Finally, by applying rule (R3), we obtain that $(q_i, n_i) \rightarrow_{\mathcal{P}'} (q_{j+1}, n_{j+1})$ by which p can be replaced, contradicting the minimality of $|\pi|$.

(3) \Rightarrow (1): We prove that $(q, n) \rightarrow_{\mathcal{P}'} (q', n)$ for some $q, q' \in Q$ implies the existence of a mountain path from (q, n) to (q', n) in $\mathfrak{S}_{\mathcal{P}}$. For proving the implication, this is sufficient since δ'_0 (resp. $\delta'_{>0}$) only differs from δ_0 (resp. $\delta_{>0}$) by adding transitions of the kind $(q, \lambda, q', 0)$. So let $n \in \mathbb{N}$ be arbitrary and let $\delta' = \delta'_0$ (resp. $\delta = \delta_0$) if $n = 0$ and $\delta' = \delta'_{>0}$ (resp. $\delta = \delta_{>0}$) if $n > 0$. We show that $(q, \lambda, q', 0) \in \delta'$ implies the existence of a mountain path from (q, n) to (q', n) in $\mathfrak{S}_{\mathcal{P}}$ by induction on the height h of the shortest proof tree for applying the rules (R1) to (R4) to deduce $(q, \lambda, q', 0) \in \delta'$. For the induction base, assume $h = 1$. Then, there are two cases. Firstly, in case we applied rule (R1), we have $(q, a, q', 0) \in \delta$ and thus clearly $(q, n) \rightarrow_{\mathcal{P}} (q', n)$ is a mountain path. Secondly, in case we applied rule (R2), we have $(q, a_1, q_1, +1) \in \delta$ and $(q_1, a_2, q', -1) \in \delta_{>0}$ for some $q_1 \in Q$ and some $a_1, a_2 \in \text{ACT}$. Hence $(q, n) \rightarrow_{\mathcal{P}} (q, n+1) \rightarrow_{\mathcal{P}} (q, n)$ is a mountain path in $\mathfrak{S}_{\mathcal{P}}$. For the induction step, firstly assume some shortest proof tree of height $h > 1$ witnessing $(q, \lambda, q', 0) \in \delta'$ has rule (R3) at its root. Then $(q, a_1, q_1, +1) \in \delta$, $(q_1, \lambda, q_2, 0) \in \delta'_{>0}$, and $(q_2, a_2, q', -1) \in \delta_{>0}$ for some $q_1, q_2 \in Q$ and some $a_1, a_2 \in \text{ACT}$. By induction hypothesis, there exists a mountain path from $(q_1, n+1)$ to $(q_2, n+1)$ in $\mathfrak{S}_{\mathcal{P}}$. Thus, in $\mathfrak{S}_{\mathcal{P}}$, we have a mountain path of the kind

$$(q, n) \rightarrow_{a_1} (q_1, n+1) \rightarrow_{\mathcal{P}} (q_2, n+1) \rightarrow_{a_2} (q', n).$$

Secondly, assume that some shortest proof tree witnessing $(q, \lambda, q', 0) \in \delta'$ has rule (R4) at its root. Then $(q, \lambda, q_1, 0) \in \delta'$ and $(q_1, \lambda, q', 0) \in \delta'$ for some $q_1 \in Q$. By induction hypothesis, there is a mountain path from (q, n) to (q_1, n) and from (q_1, n) to (q', n) in $\mathfrak{S}_{\mathcal{P}}$. Hence, there is a mountain path from (q, n) to (q', n) in $\mathfrak{S}_{\mathcal{P}}$.

D.3 Proof of Lemma 8.3.3

By definition, $\nabla(q_1, q_2)$ equals the minimal $d+1$ such that $d \in \Delta_{\downarrow}^{>0}(q_1, q_3) \cap \Delta_{\uparrow}^{>0}(q_3, q_2)$ for some $q_3 \in Q$. By applying Lemma 8.3.2, we compute in polynomial time for each $q_3 \in Q$ the two unions arithmetic progressions $\bigcup \{a_i + b_i \mathbb{N} \mid i \in [k]\}$ (resp. $\bigcup \{c_i + d_i \mathbb{N} \mid i \in [l]\}$) that equal $\Delta_{\downarrow}^{>0}(q_1, q_3)$ and (resp. $\Delta_{\uparrow}^{>0}(q_3, q_2)$), where moreover with $k, l \in O(|Q|^2)$, $a_i, c_j \in O(|Q|^2)$, and $b_i, d_j \in O(|Q|)$ for each $i \in [k]$ and each $j \in [l]$. Define $a = \max\{a_i \mid i \in [k]\}$, $b = \max\{b_i \mid i \in [k]\}$, $c = \max\{c_i \mid i \in [l]\}$, and $d = \max\{d_i \mid i \in [l]\}$. Thus, for each q_3 , it boils down to computing

$$\min \bigcup \{a_i + b_i \mathbb{N} \mid i \in [k]\} \cap \bigcup \{c_i + d_i \mathbb{N} \mid i \in [l]\}$$

which is easy, since it either equals ∞ or it is less than or equal to

$$\max\{a, c\} + b \cdot d$$

and hence is bounded by $O(|Q|^2)$.

D.4 Proof of Theorem 8.4.3

We shall now prove that, for every fixed one-counter process $\mathcal{P} = (Q, \delta_0, \delta_{>0})$, the following problem is in P: given a state $(q, n) \in Q \times \mathbb{N}$, where n is given in binary and an EF dag-formula φ , decide whether $(\mathfrak{S}_{\mathcal{P}}, (q, n)) \models \varphi$.

Assume $Q = \{q_1, \dots, q_k\}$ and $\varphi = (\varphi_i)_{i \in [l]}$. We directly refer to the translation presented earlier in this section that allows us to compute in polynomial time an extended MMA dag-formula $\alpha = (\alpha_{(i,j)})_{i \in [k] \times [l]}$ such that $\llbracket \alpha_{(i,j)} \rrbracket = \{n \in \mathbb{N} \mid (\mathfrak{S}_{\mathcal{P}}, (q_i, n)) \models \varphi_j\}$. Note that in our translation, we liberately allowed the definitions $\alpha_{(i,j)}$ to be complex. It is straightforward to see that we can compute an equivalent formula $\beta = (\beta_i)_{i \in [r]}$, where the definitions β_i are not complex along with a mapping $\varphi : [k] \times [l] \rightarrow [r]$ such that $\llbracket \alpha_{(i,j)} \rrbracket = \llbracket \beta_{\varphi(i,j)} \rrbracket$ for each $(i, j) \in [k] \times [l]$. Firstly, let us estimate r . For this, we look at the translation from \mathcal{P} and φ to α more carefully. It will suffice to estimate the definitions $\alpha_{(i,j)}$ when $\varphi_j = \text{EF} \varphi_{j'}$ for some $j' \in [j-1]$. A simple analysis

shows that rewriting the complex definition $\alpha_{(i,j)}$ in terms of non-complex definitions requires an extended MMA formula of *length* (not size) at most the product of the following

- $O(k)$ (disjunction over all $i' \in k$),
- $O(k)$ (disjunction over all $q'' \in Q$ for each $\beta_s(i, i'), s = 1, 2, 3$), and
- $O(k^3)$ (disjunction over all arithmetic progressions in Δ -sets determined either by (q_i, q'') or by $(q'', q_{i'})$ with offsets bounded by $O(k^2)$ and periods bounded by $O(k)$).

Since there are $k \cdot l$ such pairs (i, j) , we obtain that $r \in O(l \cdot k^6)$. One can easily observe from the translation that all natural numbers n such that either $x \equiv m \pmod n$ or $x \sim n$ occurs in any definition of β is bounded by $O(k^2)$, since it is at most n_{∇} plus the maximal offset of any arithmetic progression that appears in any of the Δ -sets. Since k is fixed, we obtain that L_i is a constant for each $i \in [r]$. Similarly, one verifies that the largest offset that occurs in β , that we denote by k_{β} , is also constantly bounded by $O(k^2)$, since it is less than or equal to the maximal offset of any arithmetic progression that appears in any of the Δ -sets. Hence there is some constant $c = c(\mathcal{P})$ such that, by Lemma 8.2.1, we can compute in polynomial time a threshold $t_i \leq i \cdot c + v_i$ and a period $p_i \leq c$ such that for all $n_1, n_2 > t_i$ the following implication holds

$$n_1 \equiv n_2 \pmod{p_i} \Rightarrow (n_1 \in \llbracket \beta_i \rrbracket \Leftrightarrow n_2 \in \llbracket \beta_i \rrbracket)$$

for each $i \in [r]$. Next, we aim at estimating t_i for each $i \in [r]$. A precise analysis of the definition of α shows that any constant that will occur in any definition in β_i is less than or equal to c plus the maximal periodicity p_j of any β_j with $j \in [i-1]$. Hence we have

$$v_i \leq \max\{t_j + c \mid j \in [i-1]\}$$

and

$$t_i \leq i \cdot c + v_i.$$

We claim that $t_i \leq i^2 \cdot c$ by induction on i . For the induction base, i.e. $i = 1$, observe

that $t_i = 0$ since $\alpha_1 = \top$. For the induction step, assume $i \geq 2$. Then, we have

$$\begin{aligned}
 t_i &\leq i \cdot c + v_i \\
 &= i \cdot c + \max\{t_j + c \mid j \in [i-1]\} \\
 &\leq i \cdot c + (i-1)^2 \cdot c + c \quad (\text{by induction hypothesis}) \\
 &= (i + i^2 - 2i + 2) \cdot c \\
 &\leq i^2 \cdot c
 \end{aligned}$$

The latter inequality holds since $i \geq 2$. Hence, since $i \in [r]$ and $r \in O(l \cdot k^6)$, we have $t_i \in O(l^2)$ and thus $t_i \in O(|\phi|^2)$. Thus, logarithmically many bits in $|\phi|$ suffice to represent the periods t_i . Moreover, recall that each period p_i is at most c . It is now straightforward to construct an alternating logspace Turing machine that checks if $(\mathfrak{S}_P, (q, n_0)) \models \phi$.

D.5 Proof of Proposition 8.6.2

We will reduce DSAT to the weak-bisimulation checking problem. Without loss of generality, we assume that each F_i is in 3-CNF and every assignment makes at least one clause of F_i true (this can be done by adding the clause $(x \vee \neg x)$ for some new variable x). Furthermore, we assume that each F_i is *not* a tautology (this actually means that some clauses of F_i do not contain two contradicting literals). We also assume that all the formulas F_i s have the same number of clauses; this can be done by duplicating clauses.

We are given an instance F_1, \dots, F_n of DSAT with variables x_1, \dots, x_n and set $Z = \{y_1, \dots, y_{m'}\}$ of variables such that F_i take only variables in $\{x_1, \dots, x_{i-1}\} \cup Z$. Suppose that $F_i = C_1^i \wedge \dots \wedge C_m^i$, where $C_j^i = (l_{j,1}^i \vee l_{j,2}^i \vee l_{j,3}^i)$ such that $l_{j,1}^i, l_{j,2}^i, l_{j,3}^i$ are literals over the variables $\{x_1, \dots, x_n\} \cup Z$. As we will use Gödel encoding technique, for each integer $l > 0$ let us define a function v_l mapping boolean formulas over the variables $\{x_1, \dots, x_n\} \cup Z$ to $\{\top, \perp\}$. If $\mathfrak{G}(l) = j_1 j_2 \dots$, then define $v_l(\phi)$ to be the truth value obtained by replacing y_i by j_i and x_i by $\sigma(x_i)$. It is easy to see that each $\sigma(x_i) = 1$ iff there exists an integer $l > 0$ such that $v_l(F_i) = \top$.

The one-counter net and the finite system that we will construct will use the action symbols $a, b, c, d, x_1, \overline{x_1}, \dots, x_m, \overline{x_m}, \tau$. Note that we abuse variable names x_i to also refer to action symbols; however, the meaning is clear from the context. In the following, a finite system $G = (S, \{\rightarrow_a : a \in \text{ACT}\})$ is also abbreviated as a tuple (S, δ_G) , where

q	$\delta_G(q, a)$	q	$\delta_G(q, b)$	q	$\delta_G(q, d)$
P_1	$\{A, \overline{A}\}$	A	$\{C\}$	C	$\{T, \overline{T}, X_i, \overline{X}_i : 1 \leq i \leq n\}$
P_2	$\{\overline{A}\}$	\overline{A}	$\{C, \overline{C}\}$	\overline{C}	$\{\overline{T}, X_i, \overline{X}_i : 1 \leq i \leq n\}$

Table D.1: Definition of δ_G on input a, b, d

$\delta_G : S \times \text{ACT} \rightarrow 2^S$ is a function such that $s \rightarrow_a s'$ iff $s' \in \delta_G(s)$. We construct a finite system $G = (S, \delta_G)$ as follows:

- $S = \{P_1, P_2, A, \overline{A}, C, \overline{C}, T, \overline{T}, D\} \cup \{X_i, \overline{X}_i : 1 \leq i \leq n\}$,
- The transition function δ_G is defined as follows. On input a, b and d , δ_G is defined in Table D.1. We also define that

$$\begin{aligned}
\delta_G(q, c) &= \{q\}, \forall q \in \{T, \overline{T}, X_i, \overline{X}_i : 1 \leq i \leq n\}, \\
\delta_G(q, \tau) &= \{D\}, \forall q \in \{T, X_i, \overline{X}_i : 1 \leq i \leq n\}, \\
\delta_G(q, x_i) &= \{P_1\}, \forall q \in \{T, \overline{T}, \overline{X}_j : 1 \leq j \leq n\} \cup \\
&\quad \{X_j : 1 \leq j \leq n, j \neq i\}, \\
\delta_G(q, \overline{x}_i) &= \{P_2\}, \forall q \in \{T, \overline{T}, X_j : 1 \leq j \leq n\} \cup \\
&\quad \{\overline{X}_j : 1 \leq j \leq n, j \neq i\}, \\
\delta_G(X_i, x_i) &= \{P_2\} \\
\delta_G(\overline{X}_i, \overline{x}_i) &= \{P_1\}
\end{aligned}$$

We construct a one-counter net $\mathcal{P} = (Q, \delta_0, \delta_{>0})$ as follows. Initially, Q , δ_0 , and $\delta_{>0}$ are empty. For every $1 \leq i \leq n$, we add all the states in

$$Q'_i = \{s_{F_i}, r_i, c_j^i : 1 \leq j \leq m\}.$$

If there is a literal y_k or $\neg y_k$ appearing in the clause C_j^i , then we add all the states in

$$\{\langle C_j^i, y_k, h \rangle : 0 \leq h < p_k\},$$

where p_h is the h th prime number. We also add the states in

$$\{X_i, \overline{X}_i : 1 \leq i \leq n\}.$$

In addition, we add an isomorphic copy G' of the above finite system G such that each state in G is renamed with an extra prime symbol (e.g. A becomes A'). Now we add the following transitions for each $1 \leq i \leq n$:

- add $(s_{F_i}, \tau, s_{F_i}, \pm 1)$ to $\delta_{>0}$
- add $(s_{F_i}, \tau, s_{F_i}, 1)$ to δ_0 ,
- add $(s_{F_i}, a, r_i, 0)$ to $\delta_{>0}$,
- add $(r_i, b, c_j^i, 0)$ to $\delta_{>0}$ for every $1 \leq j \leq m$,
- if $q \in \{\overline{T}', X_i', \overline{X}_i' : 1 \leq i \leq n\}$, then we add the transition $(c_j^i, d, q, 0)$ to $\delta_{>0}$,
- if the literal x_k (resp. $\neg x_k$) appears in the clause C_j^i , then we add the transition $(c_j^i, d, X_k, 0)$ to (resp. $(c_j^i, d, \overline{X}_k, 0)$) to $\delta_{>0}$,
- if the literal y_k (resp. $\neg y_k$) appears in the clause C_j^i , then we add the transition $(c_j^i, d, \langle C_j^i, y_k, 0 \rangle, 0)$ to $\delta_{>0}$,
- add $(\langle C_j^i, y_k, h \rangle, c, \langle C_j^i, y_k, h \rangle, 0)$ to $\delta_{>0}$ for all j, k, h ,
- add $(\langle C_j^i, y_k, h \rangle, \tau, \langle C_j^i, y_k, (h-1) \bmod p_k \rangle, -1)$ to $\delta_{>0}$ for all j, k, h ,
- if y_k does *not* appear positively in C_j^i , then add $(\langle C_j^i, y_k, 0 \rangle, c, \langle C_j^i, y_k, 0 \rangle, 0)$ to δ_0 ,
- if y_k does not appear negatively in C_j^i , then add $(\langle C_j^i, y_k, h \rangle, c, \langle C_j^i, y_k, h \rangle, 0)$ to δ_0 for every $1 \leq h < p_k$
- add $(\langle C_j^i, y_k, h \rangle, x_{k'}, P'_1, 0)$ for each $k' \in [n]$ to $\delta_{>0}$,
- add $(\langle C_j^i, y_k, h \rangle, \overline{x}_{k'}, P'_2, 0)$ for each $k' \in [n]$ to $\delta_{>0}$,
- if $(\langle C_j^i, y_k, h \rangle, 0)$ has a self-loop with action label c , then add $(\langle C_j^i, y_k, h \rangle, x_{k'}, P'_1, 0)$ and $(\langle C_j^i, y_k, h \rangle, \overline{x}_{k'}, P'_2, 0)$ for each $k' \in [n]$ to δ_0 ,
- add $(X_k, x_j, P'_1, 0)$ to $\delta_{>0}$ for all $j \neq k$, add $(X_k, x_k, s_{F_k}, 0)$ to $\delta_{>0}$, and add the rule $(X_k, \overline{x}_j, P'_2, 0)$ to $\delta_{>0}$ for all $0 \leq j, k \leq n$,
- add $(\overline{X}_k, x_j, P'_1, 0)$ to $\delta_{>0}$ for all $0 \leq j, k \leq n$, add $(\overline{X}_k, \overline{x}_j, P'_2, 0)$ to $\delta_{>0}$ for all $j \neq k$, and add $(\overline{X}_k, \overline{x}_k, s_{F_k}, 0)$ to $\delta_{>0}$.
- add all the transitions in G' to $\delta_{>0}$ and δ_0 , which are interpreted as internal transitions.

We claim that $\langle F_1, \dots, F_n \rangle \in \text{DSAT}$ iff $P_1 \approx s_{F_n}(0)$ iff P_2 is not weakly bisimilar to $s_{F_n}(0)$. We prove a stronger version of this claim. In the following, we define s_{F_0} to be the state P'_1 , and use the convention that the empty sequence $\langle \rangle$ of formulas is an instance of DSAT.

Claim D.5.1 *For every $0 \leq i \leq n$, the following statements are equivalent:*

- $\langle F_1, \dots, F_i \rangle \in \text{DSAT}$.
- $P_1 \approx s_{F_i}(l)$ for all $l \in \mathbb{N}$.
- P_2 is not weakly bisimilar to $s_{F_i}(l)$ for all l in \mathbb{N} .

We prove this claim by induction on i . Consider the base case $i = 0$. Our convention implies that $\langle \rangle \in \text{DSAT}$. Observe also that each state in G is obviously bisimilar to the corresponding state in G' after renaming with any given counter value (e.g. $A \approx A'(j)$ for any $j \in \mathbb{N}$). This is because no states in G' ever reach a state that modifies the counter values. In particular, we have $P_1 \approx P'_1(0) = s_{F_0}(0)$. Furthermore, it is easy to see that P_1 is not weakly bisimilar to P_2 . This also means that P_2 is not weakly bisimilar to $s_{F_0}(0)$.

We now consider the inductive case $i > 0$. We shall use the function v_l that we defined earlier in the proof. We now give several obvious facts (one easily follows from the previous ones):

- For each $l > 0$, if the variable y_k appears in the clause C_j^i , then $\langle C_j^i, y_k, 0 \rangle(l) \xrightarrow{\tau} \langle C_j^i, y_k, h \rangle(0)$ iff $l \equiv h \pmod{p_k}$.
- For each $1 \leq j < i$, it is the case that $T \approx X_j(l)$ for each $l > 0$ iff the unique assignment $\sigma : \{x_1, \dots, x_n\} \rightarrow \{0, 1\}$ for $\langle F_1, \dots, F_n \rangle$ satisfies $\sigma(x_j) = 1$. To see this, observe first that Attacker cannot start the game by playing any moves with action labels other than x_j , which will take a pebble to either G or G' , as Defender then has a response in the isomorphic copy in the other system. So, Attacker is then forced to make x_j -move in either of the system, which does not matter as both of the states X_j and $X_j(l)$ has exactly one unique x_j -successor. Defender has a unique response which set up the next configuration $\langle P_1, s_{F_j}(l) \rangle$. By induction, Defender now wins iff $P_1 \approx s_{F_j}(l)$ iff $\sigma(x_j) = 1$.
- For each $1 \leq j < i$, it is the case that $T \approx \overline{X_j}(l)$ for each $l > 0$ iff $\sigma(x_j) = 0$. The proof is identical to the previous item.

- $v_l(C_j^i) = \top$ iff $C \approx c_j^i(l)$. To see this, observe first that Attacker cannot make a d -move to any of the states in $\{\overline{T}, X_k, \overline{X}_k : 1 \leq k \leq n\}$ in G or $\{\overline{T}', X'_k, \overline{X}'_k : 1 \leq k \leq n\}$ in G' , as Defender then has a response in the isomorphic copy in the other system. Attacker then has only two choices: (1) a d -move to T in G , or (2) a d -move to the states in S corresponding to the literals in C_j^i . Suppose that $C \approx c_j^i(l)$. Then, Defender has a response in S for Attacker's move of type (1): either one of the state $\langle C_j^i, y_k, h \rangle(l')$, or a state X_k , or a state \overline{X}_k . For the first case, there is a literal $(\neg)y_k$ that makes C_j^i true under v_l . For the second (resp. third) case, x_k (resp. $\neg x_k$) is a literal in C_j^i with $\sigma(x_k) = 1$ (resp. $\sigma(x_k) = 0$). Conversely, suppose that $v_l(C_j^i) = \top$. We shall prove that $C \approx c_j^i(l)$. For Attacker's move of type (1), Defender has a winning strategy that is similar to the converse case. Let us consider Attacker's move of type (2). Suppose that a variable y_k appears in C_j^i and Attacker makes a d -move to $\langle C_j^i, y_k, 0 \rangle$. Suppose first that y_k does not appear positively in C_j^i . If $v_l(\neg y_k) = \perp$, Defender plays a d -move to \overline{T} ; and if $v_l(\neg y_k) = \top$, Defender plays a d -move to T . Similarly, Defender has a response when y_k does not appear negatively in C_j^i . Suppose now that a literal x_k appears in C_j^i and Attacker makes a d -move to X_k . If $\sigma(x_k) = 1$, it is easy to see that Defender can make a d -move to T and wins. If $\sigma(x_k) = 0$, Defender can make a d -move to X_k in G and wins since the unique x_k -successor from X_k is P_2 instead of P_1 (i.e. state X_k “inverts” the outcome of the state T under the action x_k). Similarly, if the literal x_k is contained in C_j^i and Attacker makes a d -move to \overline{X}_k , then Defender has a winning response.
- $v_l(C_j^i) = \perp$ iff $\overline{C} \approx c_j^i(l)$. The proof is similar to the previous case.
- $v_l(F_i) = \top$ iff $A \approx r_i(l)$. To see this, observe that $v_l(F_i) = \top$ iff $v_l(C_j^i) = \top$ for all j . By the previous items, this in turn is true iff $C \approx c_j^i(l)$ for all j .
- $v_l(F_i) = \perp$ iff $\overline{A} \approx r_i(l)$. This can be proved in the same way as in the previous item. Here, we need the extra assumption that every assignment makes at least one clause of F_i true so that the transition $\overline{A} \rightarrow_a C$ can be matched in $r_i(l)$.

Finally, it is easy to deduce the above Claim from the last two items. This is because we assumed that none of the formulas in F_1, \dots, F_n is a tautology.

D.6 Proof of Proposition 8.6.3

To prove this lower bound, we will make use of an algorithm given by Kučera [Kuc00] as a subroutine:

Lemma.([Kuc00]) *There exists a fixed finite system \mathcal{F}' with states P_1 and P_2 such that, given a boolean formula ϕ , we can compute in P a one-counter net \mathfrak{D} and an initial configuration $(q_0, 0)$ such that $(q_0, 0) \approx P_1$ iff ϕ is satisfiable, and $(q_0, 0) \approx P_2$ iff ϕ is unsatisfiable.*

The proof is by a polynomial time reduction from the problem INDEX-ODD: given a list F_1, \dots, F_n of boolean formulas in 3-CNF, does there exist an odd index $1 \leq i \leq n$ such that F_1, \dots, F_i are all satisfiable and F_{i+1}, \dots, F_n are all unsatisfiable? This problem is $P^{NP[\log]}$ -complete. A $P^{NP[\log]}$ upper bound is immediate by a simple binary search in the list F_1, \dots, F_n by invoking an NP oracle at each step to determine the rightmost satisfiable formula F_i . A $P^{NP[\log]}$ -hardness for INDEX-ODD is also immediate from Wagner's sufficient conditions for $P^{NP[\log]}$ -hardness [Wag87, Theorem 5.2] (see also [SV00, Lemma 7]). For notational convenience, we assume that INDEX-ODD accepts only lists of formulas ψ_1, \dots, ψ_n satisfying the following extra restrictions: (1) $n \geq 3$ is odd, and (2) ψ_n is unsatisfiable. It is easy to see that the problem remains $P^{NP[\log]}$ -complete, e.g., by adding at most two extra formulas \perp to the end of the input list of formulas. Therefore, we may assume that the given input is the list $\psi_1, \dots, \psi_{2k}, \psi_{2k+1}$, where $\psi_{2k+1} = \perp$. Initially, we run Kučera's algorithm sequentially on inputs ψ_1, \dots, ψ_{2k} to obtain the one-counter nets $\mathcal{P}_1, \dots, \mathcal{P}_{2k+1}$ with respective initial states q_1, \dots, q_{2k+1} such that, for all $i \in [2k+1]$, the following conditions hold for the states P_1 and P_2 in the finite system $\mathcal{F}' = (Q_{F'}, \delta_{F'})$ given by the above lemma.

1. $(q_i, 0) \approx P_1$ iff ψ_i is satisfiable (*)
2. $(q_i, 0) \approx P_2$ iff ψ_i is unsatisfiable (**)

We will now define a one-counter net $\mathcal{P} = (Q, \delta_{>0}, \delta_0)$ and a fixed finite system $\mathcal{F} = (S, \delta_F)$. We start with the definition of \mathcal{F} . We define $S = \{t_0\} \cup \{t_{i,j} : i, j \in [3]\} \cup Q_{F'}$. Intuitively, if we pick an index $h \in [2k-1]$, then obviously the formulas ψ_1, \dots, ψ_h are either: (1) all satisfiable, (2) all unsatisfiable, or (3) some are satisfiable and the rest are unsatisfiable. These three split cases correspond to the three possible left indices i for $t_{i,j}$. We have also three similar split cases if we inspect the formulas $\psi_{h+1}, \dots, \psi_{2k+1}$.

[Actually, one of these cases are impossible as we always assume that ψ_{2k+1} is unsatisfiable.] Hence, the right indices j of $t_{i,j}$ similarly correspond to the these three possible split cases. We now define the transitions for \mathcal{F} accordingly:

- add each transition $(t_0, e, t_{i,j})$ to δ_F for each $i, j \in [3]$.
- add each transition $(t_{i,j}, L, P_1)$ to δ_F for each $i \in \{1, 3\}$ and $j \in [3]$.
- add each transition $(t_{i,j}, L, P_2)$ to δ_F for each $i \in \{2, 3\}$ and $j \in [3]$.
- add each transition $(t_{i,j}, R, P_1)$ to δ_F for each $i \in [3]$ and $j \in \{1, 3\}$.
- add each transition $(t_{i,j}, R, P_2)$ to δ_F for each $i \in [3]$ and $j \in \{2, 3\}$.
- add each transition in $\delta_{F'}$ to δ_F .

We continue now with the definition of \mathcal{P} . We assume that \mathcal{P} initially has neither states nor transitions, and continue adding states/transitions as follows:

- Add the initial state q_0 and the states in $\{s_i : 1 \leq i < 2k \text{ and } i \text{ is odd}\}$.
- Add the transition $(q_0, e, s_i, 0)$ to $\delta_{>0}$ and δ_0 for each s_i ,
- Assume that the states of $\mathcal{P}_1, \dots, \mathcal{P}_{2k+1}$ are disjoint and add the states and transitions of these one-counter nets into \mathcal{P} .
- Add the transition $(s_i, L, q_j, 0)$ to $\delta_{>0}$ and δ_0 for each s_i and $j \in [i]$.
- Add the transition $(s_i, R, q_j, 0)$ to $\delta_{>0}$ and δ_0 for each s_i and $2k+1 \geq j > i$.
- Add an isomorphic copy \mathcal{S} of \mathcal{F} into \mathcal{P} ; each new state in \mathcal{S} is renamed with an extra prime symbol so as to distinguish from the states of \mathcal{F} (e.g. $t_{1,2}$ in \mathcal{F} is renamed to $t'_{1,2}$ in \mathcal{S}).
- Add each transition $(q_0, e, t_{i,j}, 0)$ to $\delta_{>0}$ and δ_0 for each $i, j \in [3]$, except for $(i, j) = (1, 2)$.

We want to prove that $(q_0, 0) \approx t_0$ iff $\psi_1, \dots, \psi_{2k+1}$ is an instance of INDEX-ODD, which suffices to deduce our theorem.

We first make a simple observation. For each $s_i \in \mathcal{Q}$, $(s_i, 0) \approx t_{1,2}$ iff the formulas ψ_1, \dots, ψ_i are all satisfiable and the formulas $\psi_{i+1}, \dots, \psi_{2k+1}$ are all unsatisfiable. To see this, note that by definition action L can only take $t_{1,2}$ in \mathcal{F} to P_1 (i.e. asserting

satisfiability), while action R can only take $t_{1,2}$ to P_2 (i.e. asserting unsatisfiability). Also, by definition, action L can only take s_i to q_j with $j \leq i$, while action R can only take s_i to q_j with $j > i$. This observation is then an immediate consequence of property (*) and (**).

We now prove that $(q_0, 0) \approx t_0$ iff there exists an odd index $i < 2k + 1$ such that ψ_1, \dots, ψ_i are all satisfiable and $\psi_{i+1}, \dots, \psi_{2k+1}$ are all unsatisfiable. This suffices to deduce our theorem. Let us first prove necessity. Assume that $(q_0, 0) \approx t_0$. Then, Attacker can move the pebble from t_0 to $t_{1,2}$. Note that there is no transition from q_0 to $t'_{1,2}$ in \mathcal{P} . This means that Defender's winning strategy must choose one of the states s_i . That is, we have $(s_i, 0) \approx t_{1,2}$ and the rest follows from the previous simple observation. Conversely, given the odd index i , we want to show that $(q_0, 0) \approx t_0$. If Attacker chooses $t_{1,2}$, then Defender can move the pebble from $(q_0, 0)$ to $(s_i, 0)$ and win as in the converse case. If Attacker chooses other $t_{i,j}$ with $(i, j) \neq (1, 2)$, then Defender can move the pebble from $(s_i, 0)$ to $(t'_{i,j}, 0)$ and win as obviously $(t'_{i,j}, l) \approx t_{i,j}$ for all $l \in \mathbb{N}$. Defender has a similar winning response if Attacker moves the pebble from $(q_0, 0)$ to $(t'_{i,j}, 0)$ with $(i, j) \neq (1, 2)$. The remaining possible moves for Attacker are moves from $(q_0, 0)$ to $(s_{i'}, 0)$ for some odd index $i' < 2k + 1$. If $i' = i$, then Defender can move from t_0 to $t_{1,2}$ and win. If $i' \neq i$, then we can look at the formulas $\psi_1, \dots, \psi_{i'}$ and the formulas $\psi_{i'+1}, \dots, \psi_{2k+1}$. Each of these lists contain either (1) all satisfiable formulas, (2) all unsatisfiable formulas, or (3) both satisfiable and unsatisfiable formulas. Considering these two lists separately, this gives rise to nine possible combinations, which are all covered by the successors of t_0 in \mathcal{F} . Therefore, by picking the appropriate successor of t_0 , Defender can win.

Appendix E

Proofs from Chapter 9

E.1 Proof of Proposition 9.2.1

We shall now sketch a reduction from model checking $\text{FO}_{\mathcal{S}}(\text{Reach})$ and $\text{FO}^2(\text{Reach})$ over ΠOCP to, respectively the logic \mathcal{L} and \mathcal{L}' . We will first deal with atomic propositions. To this end, we will state a lemma, whose proof immediately follows from Lemma 8.3.1, Lemma 8.3.4, and Lemma 8.3.5.

Lemma E.1.1 *Given an OCP $\mathcal{P} = (Q, \delta_0, \delta_{>0})$ over ACT with $Q = \{q_0, \dots, q_k\}$ and a subset $\text{ACT}' \subseteq \text{ACT}$, one can compute in poly-time a quantifier-free \mathcal{L}' -formula $\varphi_{\text{ACT}'}(x_1, x_2, y_1, y_2)$ in disjunctive normal form (DNF) such that for all $a_1, a_2, b_1, b_2 \in \mathbb{N}$,*

$$\mathbb{N} \models \varphi(a_1, a_2, b_1, b_2) \Leftrightarrow (q_{a_1}, a_2) \rightarrow_{\text{ACT}'}^* (q_{b_1}, b_2).$$

Although Lemma 8.3.1, Lemma 8.3.4, and Lemma 8.3.5 discuss only the reachability relation \rightarrow^* without any labeling constraints, this lemma can be obtained by first removing transitions with labels in $\text{ACT} - \text{ACT}'$ from the given OCP (in the same way we deal with $\text{EF}_{\text{ACT}'}$ in the proof of Theorem 8.4.1). Note that formulas in DNF are of alternation rank 2. Since the synchronization constraints $=_{i,j}$ over transition systems of the form $\mathcal{S}_{\mathcal{P}}^S$ introduce only self-loops, the following lemma can be directly deduced from this by taking a conjunction, which increases the alternation rank by 1.

Lemma E.1.2 *Let \mathcal{P} be an asynchronous product of r OCPs $\mathcal{P}_1, \dots, \mathcal{P}_r$. Let $Q = \{q_0, \dots, q_k\}$ be the union of their control states. Suppose that $\mathcal{S}_{\mathcal{P}}^S$ has action labels ACT. Then, for each $\text{ACT}' \subseteq \text{ACT}$, one can compute in poly-time a quantifier-free \mathcal{L}' -formula $\varphi_{\text{ACT}'}(\bar{x}, \bar{y})$ with alternation rank 3, where $\bar{x} = (x_1, \dots, x_{2r})$ and $\bar{y} =$*

(y_1, \dots, y_{2r}) , such that for all two tuples $\bar{a} = (a_1, \dots, a_{2r})$ and $\bar{b} = (b_1, \dots, b_{2r})$ of natural numbers, it is the case that

$$\langle \mathbb{N}, + \rangle \models \Phi_{ACT'}(\bar{a}, \bar{b}) \Leftrightarrow ((q_{a_1}, a_2), \dots, (q_{a_{2r-1}}, a_{2r})) \rightarrow_{\Sigma'}^* ((q_{b_1}, b_2), \dots, (q_{b_{2r-1}}, b_{2r}))$$

in the transition system $\mathfrak{S}_{\mathcal{P}}^S$.

This lemma shows that atomic propositions of the form $\text{Reach}_{ACT'}(x, y)$ can be dealt with. Furthermore, it is not hard to give a quantifier-free \mathcal{L}' -formula $\phi_a(\bar{x}, \bar{y})$ (resp. $\phi_{=i,j}(\bar{x}, \bar{y})$) with alternation rank at most 3 representing the atomic propositions of the form $E_a(x, y)$ (resp. $x =_{i,j} y$). For example, for formulas of the form $E_a(x, y)$, one simply needs to check only local transitions in each OCP and distinguish the case where the initial counter value is zero or not, all of which can be encoded in \mathcal{L} as a big disjunction of these cases. An extra big conjunction in the outermost layer is needed to encode the asynchronous product.

The inductive cases can be dealt with easily. The case of boolean combinations is immediate. An existential quantifier for an $\text{FO}_{\mathcal{S}}(\text{Reach})$ (resp. $\text{FO}^2(\text{Reach})$) can be replaced by a block of r existential quantifiers, where r is the number of OCPs in the given asynchronous product. Finally, it is easy to see that the alternating rank of the output formula equals the alternating rank of the input formula up to an addition by a small constant factor c (from our proof, $c = 3$).

E.2 Missing proofs from Subsection 9.3.2

Definition E.2.1 Given $i \in \mathbb{N}$, $p, m \in \mathbb{Z}_{>0}$, and two $2n$ -tuples $(\bar{a}, \bar{b}), (\bar{c}, \bar{d}) \in \mathbb{N}^{2n}$, where $\bar{a} = (a_1, \dots, a_n)$, $\bar{b} = (b_1, \dots, b_n)$, $\bar{c} = (c_1, \dots, c_n)$, and $\bar{d} = (d_1, \dots, d_n)$, we write

$$(\bar{a}, \bar{b}) \sim_{p,m}^i (\bar{c}, \bar{d})$$

iff for all $j \in [1, n]$ the following statements hold

1. $d(a_j, b_j) \leq pm$ implies $d(a_j, b_j) = d(c_j, d_j)$,
2. $d(c_j, d_j) \leq pm$ implies $d(a_j, b_j) = d(c_j, d_j)$,
3. $d(a_j, b_j) > pm$ iff $d(c_j, d_j) > pm$,
4. $a_j \leq 2(i+1)pm$ implies $a_j = c_j$,
5. $b_j \leq 2(i+1)pm$ implies $b_j = d_j$,

6. $c_j \leq 2(i+1)pm$ implies $a_j = c_j$,
7. $d_j \leq 2(i+1)pm$ implies $b_j = d_j$,
8. $a_j \equiv c_j \pmod{p}$ and $b_j \equiv d_j \pmod{p}$, and
9. $a_j \leq b_j$ iff $c_j \leq d_j$.

From this definition, it follows that $(\bar{a}, \bar{b}) \sim_{p,m}^i (\bar{c}, \bar{d})$ iff, for each $j \in [1, n]$, it is the case that $(a_j, b_j) \sim_{p,m}^i (c_j, d_j)$.

Lemma E.2.1 Suppose $i \in \mathbb{Z}_{>0}$ and $\bar{a}, \bar{b}, \bar{c}, \bar{d} \in \mathbb{N}^n$. If $(\bar{a}, \bar{b}) \sim_{p,m}^i (\bar{c}, \bar{d})$, then the following statements hold:

1. For all $\bar{a}' \in \mathbb{N}^n$, there exists $\bar{c}' \in \mathbb{N}^n$ such that $(\bar{a}', \bar{b}) \sim_{p,m}^{i-1} (\bar{c}', \bar{d})$.
2. For all $\bar{b}' \in \mathbb{N}^n$, there exists $\bar{d}' \in \mathbb{N}^n$ such that $(\bar{a}, \bar{b}') \sim_{p,m}^{i-1} (\bar{c}, \bar{d}')$.

Proof. We prove the first statement; the second statement can be proved in the same way. Suppose that $\bar{a}' = (a'_1, \dots, a'_n)$. For each $j \in [1, n]$ we shall define $c'_j \in \mathbb{N}$ satisfying $(a'_j, b_j) \sim_{p,m}^{i-1} (c'_j, d_j)$. This suffices for showing $(\bar{a}', \bar{b}) \sim_{p,m}^{i-1} (\bar{c}', \bar{d})$. Furthermore, c' will be at most $\max(\bar{d}) + 2pm$. There are two possibilities:

- Either $b_j \leq 2(i+1)pm$ or $d_j \leq 2(i+1)pm$ holds. In this case, our assumption that $(\bar{a}, \bar{b}) \sim_{p,m}^i (\bar{c}, \bar{d})$ implies that $b_j = d_j$. If $a'_j \leq 2(i+2)pm$, then choose $c'_j = a'_j$, which would show that $(a'_j, b_j) \sim_{p,m}^{i-1} (c'_j, d_j)$. Otherwise, we have $a'_j > 2(i+2)pm$ and choose

$$c'_j = \begin{cases} 2(i+1)pm + pm + (a'_j \bmod p) & , \text{ if } p \nmid a'_j \\ 2(i+1)pm + pm + p & , \text{ if } p \mid a'_j. \end{cases}$$

Then, it is easy to check that $(a'_j, b_j) \sim_{p,m}^{i-1} (c'_j, d_j)$. This also shows that our choice of c'_j does not have to exceed $2(i+1)pm + 2pm$.

- Both $b_j > 2(i+1)pm$ and $d_j > 2(i+1)pm$ hold. First consider the case when $d(a'_j, b_j) \leq pm$. In this case, choose $c'_j := d_j + (a'_j - b_j)$. Observe that $c'_j > 2ipm$, and we have $a_j \leq b_j \Leftrightarrow c_j \leq d_j$. Furthermore, using our assumption $(\bar{a}, \bar{b}) \sim_{p,m}^i (\bar{c}, \bar{d})$, it is easy to use check that $(a'_j, b_j) \sim_{p,m}^{i-1} (c'_j, d_j)$. Consider now the case when $d(a'_j, b_j) > pm$. There are several further possibilities:

- $a'_j \leq 2ipm$. Choose $c'_j = a'_j$ and so we have $d(a'_j, b_j), d(c'_j, d_j) > pm$. Together with our assumption that $(\bar{a}, \bar{b}) \sim_{p,m}^i (\bar{c}, \bar{d})$, we have $(a'_j, b_j) \sim_{p,m}^{i-1} (c'_j, d_j)$.
- $a'_j > 2ipm$. In this case, if $a'_j < b_j$, we choose $c'_j := d_j - p(m+1) + [(a'_j - b_j) \bmod p]$. We have $c'_j > 2(i+1)pm - p(m+1) \geq 2ipm$ and $d(c'_j, d_j) > pm$. It is easy now to check that $(a'_j, b_j) \sim_{p,m}^{i-1} (c'_j, d_j)$. On the other hand, if $a'_j > b_j$, we choose $c'_j := d_j + p(m+1) - [(b_j - a'_j) \bmod p]$. It follows that $d(c'_j, d_j) > pm$. It is easy to check that $(a'_j, b_j) \sim_{p,m}^{i-1} (c'_j, d_j)$.

□

Suppose $i > 0$. Let C and C' be, respectively, a $\sim_{p,m}^i$ -equivalence class and a $\sim_{p,m}^{i-1}$ -equivalence class. We say that C' is *x-consistent with C* if there exist tuples $\bar{a}, \bar{b}, \bar{a}' \in \mathbb{N}$ such that $(\bar{a}, \bar{b}) \in C$ and $(\bar{a}', \bar{b}) \in C'$. Similarly, C' is *y-consistent with C* if there exist tuples $\bar{a}, \bar{b}, \bar{b}' \in \mathbb{N}$ such that $(\bar{a}, \bar{b}) \in C$ and $(\bar{a}, \bar{b}') \in C'$.

Lemma E.2.2 *Let $\bar{a}, \bar{b} \in \mathbb{N}^n$ and C be the $\sim_{p,m}^i$ -equivalence class of (\bar{a}, \bar{b}) . Define*

$$\begin{aligned} A &:= \{(a'_1, a'_2, \dots, a'_n, \bar{b}) : \forall j \in [1, n] \ 0 \leq a'_j \leq \max(\bar{b}) + 2pm\} \\ B &:= \{(\bar{a}, b'_1, b'_2, \dots, b'_n) : \forall j \in [1, n] \ 0 \leq b'_j \leq \max(\bar{a}) + 2pm\} \end{aligned}$$

Then, the set A (resp. B) contains a representative of every $\sim_{p,m}^{i-1}$ -equivalence class x-consistent (resp. y-consistent) with C .

Proof. This follows from the proof of Lemma E.2.1, where c'_j (resp. d'_j) was always chosen to be at most $\max(\bar{d}) + 2pm$ (resp. $\max(\bar{c}) + 2pm$). □

Lemma E.2.3 *Let $p, m, n \in \mathbb{Z}_{>0}$ and $k \in \mathbb{N}$. Let $\bar{a}, \bar{b}, \bar{c}, \bar{d} \in \mathbb{N}^n$. If $(\bar{a}, \bar{b}) \sim_{p,m}^k (\bar{c}, \bar{d})$, then for each formula $\varphi \in \mathcal{L}'_{p,m}(n)$ with quantifier rank at most k we have*

$$\langle \mathbb{N}, + \rangle \models \varphi(\bar{a}, \bar{b}) \Leftrightarrow \langle \mathbb{N}, + \rangle \models \varphi(\bar{c}, \bar{d}).$$

Proof. The proof is by induction on φ . First consider the base cases:

- $x_j \sim y_j + c$ and $y_j \sim x_j + c$ for $\sim \in \{\leq, \geq, =\}$. Same as in the proof of Lemma 9.3.5.

- $x_j \sim c$. If $a_j \leq 2(i+1)pm$ or $c_j \leq 2(i+1)pm$, then $a_j \sim_{p,m}^k c_j$ implies that $a_j = c_j$, and so $a_j \sim c$ iff $c_j \sim c$. If $a_j > 2(i+1)pm$ and $c_j > 2(i+1)pm$, then we also obviously have $a_j \sim c$ iff $c_j \sim c$ since $c \leq m < 2(i+1)pm$.
- $y_j \sim c$. Same as previous item.
- $x_j \equiv y_j + c \pmod{d}$, $y_j \equiv x_j + c \pmod{d}$, $x_j \equiv c \pmod{d}$, and $y_j \equiv c \pmod{d}$. Same as in the proof of Lemma 9.3.5.

We now turn to the inductive cases. Boolean combinations are easy. So, let ϕ be a formula of the form $\exists x_j \psi$. We now prove that $\langle \mathbb{N}, + \rangle \models \phi(\bar{a}, \bar{b})$ implies $\langle \mathbb{N}, + \rangle \models \phi(\bar{c}, \bar{d})$; the converse is symmetric. If $\langle \mathbb{N}, + \rangle \models \phi(\bar{a}, \bar{b})$, then there exists $a'_j \in \mathbb{N}$ such that $\langle \mathbb{N}, + \rangle \models \phi(\bar{a}', \bar{b})$, where $\bar{a}' = (a_1, \dots, a_{j-1}, a'_j, a_{j+1}, \dots, a_n)$. Since $(\bar{a}, \bar{b}) \sim_{p,m}^k (\bar{c}, \bar{d})$, Lemma E.2.1 now implies that there exists c'_j such that, whenever

$$\bar{c}' = (c_1, \dots, c_{j-1}, c'_j, c_{j+1}, \dots, c_n),$$

we have $(\bar{a}', \bar{b}) \sim_{p,m}^{k-1} (\bar{c}', \bar{d})$. The quantifier rank of ψ is at most $k-1$ and the induction hypothesis implies $\langle \mathbb{N}, + \rangle \models \psi(\bar{c}', \bar{d})$, which in turn means $\langle \mathbb{N}, + \rangle \models \phi(\bar{c}, \bar{d})$. If ϕ is of the form $\exists y_j \psi$, the proof is also the same. \square

Lemma E.2.4 *Let $\bar{a}, \bar{b} \in \mathbb{N}^n$. Then, there exists $\bar{c}, \bar{d} \in [0, 2(i+3)pm]^n$ such that*

$$(\bar{a}, \bar{b}) \sim_{p,m}^i (\bar{c}, \bar{d}).$$

Furthermore, for fixed $p, m, n > 0$, on inputs \bar{a} and \bar{b} (represented in binary), and i (represented in unary), a logspace machine can compute \bar{c} and \bar{d} and store them in its working tape.

Proof. We first show the existence of \bar{c} and \bar{d} . For this, it suffices to show that, for each $j \in [1, n]$, there exists $c_j, d_j \in [0, 2(i+3)pm]$ such that $(a_j, b_j) \sim_{p,m}^i (c_j, d_j)$. It suffices to consider the case where either $a_j > 2(i+3)pm$ or $b_j > 2(i+3)pm$ holds. If $a_j = b_j$, then it is easy to show that setting $c_j = d_j = 2(i+1)pm + p + (a_j \bmod p)$ satisfies $(a_j, b_j) \sim_{p,m}^i (c_j, d_j)$. Now consider the case when $a_j < b_j$; the case when $a_j > b_j$ is similar.

First, consider the case $d(a_j, b_j) \leq pm$. In this case, we have $a_j > 2(i+1)pm$; for, otherwise, $b_j = a_j + (b_j - a_j) \leq 2(i+1)pm + pm < 2(i+3)pm$, contradicting our assumption. Therefore, we set $c_j = 2(i+1)pm + p + (a_j \bmod p)$ and $d_j = 2(i+1)pm + p + (a_j \bmod p) + (b_j - a_j)$. It is easy to check that $c_j, d_j \in [0, 2(i+3)pm]$ and $(a_j, b_j) \sim_{p,m}^i (c_j, d_j)$.

Now consider the case when $d(a_j, b_j) > pm$. If $a_j \leq 2(i+1)pm$, then our assumption implies that $b_j > 2(i+3)pm$. So, we set $c_j = a_j$ and $d_j = 2(i+1)pm + pm + p + (b_j \bmod p)$. It is easy to see that $d_j \in [0, 2(i+3)pm]$ and $(a_j, b_j) \sim_{p,m}^i (c_j, d_j)$. Finally, if $a_j > 2(i+1)pm$, we set $c_j = 2(i+1)pm + p + (a_j \bmod p)$ and $d_j = c_j + pm + p - [(a_j - b_j) \bmod p]$. This means that $d_j \leq 2(i+1)pm + 3p + pm \leq 2(i+3)pm$. Since it is easy to check that $a_j \equiv c_j \pmod{p}$, it follows that $b_j \equiv d_j \pmod{p}$. Furthermore, it is easy to check that $(a_j, b_j) \sim_{p,m}^i (c_j, d_j)$.

To show that there is a logspace machine computing \bar{c} and \bar{d} and store them in the working tape, observe first that fixing the parameters p, m, n means that the size of binary representations of \bar{c} and \bar{d} is $O(\log |i|)$, since i is represented in unary, and so can be stored in the working tape of a logspace machine. To see that one requires only logspace for actual the computation, recall that checking whether $s|t$, given the numbers s and t as inputs represented in binary, can be done in L (e.g. see the survey [All01]). Therefore, since p is fixed, to compute the number $(s \bmod p)$ one can simply sequentially go through $j = 0, \dots, p-1$ and check whether $p|(s-j)$. Finally, observe that the rest of the arithmetic operations above (i.e. additions, multiplications, and (in)equality tests) can easily be done by a logspace machine. So, the proof can be directly translated into the desired logspace machine. \square

Proof of Proposition 9.3.7. We give an alternating logspace Turing machine M for solving the membership problem of $\mathcal{L}'_{p,m}(n)$. Given $\bar{a}, \bar{b} \in \mathbb{N}^n$ and a formula $\phi(\bar{x}, \bar{y}) \in \mathcal{L}'_{p,m}(n)$ as input, note that the quantifier rank of ϕ is bounded above by $i := \|\phi\|$, i.e., the number i is represented in unary on the input tape. So, using Lemma E.2.4 our machine computes $\bar{c}, \bar{d} \in [0, 2(i+3)pm]^n$ such that $(\bar{a}, \bar{b}) \sim_{p,m}^i (\bar{c}, \bar{d})$. Therefore, by Lemma E.2.3, M needs only check whether $\langle \mathbb{N}, + \rangle \models \phi(\bar{c}, \bar{d})$. The machine M checks this by a (single) one-way top-down traversal of the parse tree of ϕ in the input tape starting at the root, which is the formula ϕ . At each step, M keeps track of a valuation \bar{e}, \bar{g} of the variables in \bar{x}, \bar{y} , where $\bar{e} = (e_1, \dots, e_n)$ and $\bar{g} = (g_1, \dots, g_n)$. Initially, \bar{e} (resp. \bar{g}) is set to \bar{c} (resp. \bar{d}). Suppose that the subformula of ϕ being traversed is α . If α is an atomic proposition, then M can check ϕ easily since all the required numbers are logarithmically bounded (or fixed). If α is of the form $\psi \vee \psi'$, then M existentially chooses the next subformula α' to be either ψ or ψ' , and checks whether $\langle \mathbb{N}, + \rangle \models \alpha'(\bar{e}, \bar{g})$. If α is of the form $\neg\psi$, then M simply makes a transition into a not-state and checks whether $\langle \mathbb{N}, + \rangle \models \psi(\bar{e}, \bar{g})$. If α is of the form $\exists x_i \psi$, then M will existentially guess a number $e'_j \leq g_j + 2pm$. Letting $\bar{e}' = (e_1, \dots, e_{j-1}, e'_j, e_{j+1}, \dots, e_n)$, the machine M then checks whether $\langle \mathbb{N}, + \rangle \models \psi(\bar{e}', \bar{g})$. The machine M needs not guess a number bigger

than $g_j + 2pm$ due to Lemma E.2.2. The case when ϕ is of the form $\exists y_j \psi$ is similar.

At each step, the machine M requires to keep track of: 1) which subformula of ϕ the machine M is currently at, for which we need $O(\log i)$ bits, and 2) the current valuations \bar{e}, \bar{g} for the variables \bar{x}, \bar{y} . Notice that at each step $\max(\bar{e}, \bar{g}) \leq \max(\bar{e}, \bar{d}) + 2ipm$, which requires $O(\log i)$ bits as well to represent. \square

E.3 Proof of Lemma 9.4.1

We first show upper bounds. Given an arena $\mathcal{A} = (\bar{v}, k, \phi)$ and two positive integers s, t , we write $s \sim_{\mathcal{A}} t$ iff $s \equiv t \pmod{\prod_{j=1}^K p_j}$, where p_K is the largest prime appearing in ϕ . Then, each equivalence class in $\sim_{\mathcal{A}}$ has a representative that is at most $\prod_{j=1}^K p_i$, whose size when represented in binary does not exceed $p(|\mathcal{A}|)$ for some fixed polynomial p not depending on the input arena (recall that each number in \mathcal{A} is given in unary). As each atomic proposition in ϕ is only a divisibility test of the form $p|x$ with $p \leq p_K$, it is easy to check that a polynomial-time alternating Turing machine M solving the buffer game need only simulate the game, while considering only the smallest representative modulo $\prod_{j=1}^K p_i$ of each equivalence classes in $\sim_{\mathcal{A}}$. For each guessing step, the machine M will ensure that the important information from the earlier round is not “overwritten”. When a representative $[m_k]$ of the last buffer value has been obtained, the machine M will check whether $\langle \mathbb{N}, + \rangle \models \phi([m_k])$. Checking these can be done in polynomial time as divisibility of numbers represented in binary can be checked in polynomial time. Finally, observe that the number of alternations used by M on input \mathcal{A} is exactly $k+1$ (starting with existential guess). This shows that BUFFER is in PSPACE, and BUFFER_k is in Σ_{k+1}^P .

To show lower bounds, we give a polynomial-time reduction from the well-known PSPACE-complete problem QBF. We assume that the leading quantifier of the input formulas is \exists . Moreover, if the input quantifier boolean formula ϕ has $k+1$ quantifier alternations, the resulting output arena is of the form (\bar{v}, k, ϕ) . In particular, this will prove that BUFFER_k is Σ_{k+1}^P -hard, as the restricted QBF problem with a fixed number $k+1$ of quantifier alternations starting with the quantifier \exists is Σ_{k+1}^P -complete. Suppose that the input is the formula

$$\theta = Q_1 x_1 \dots Q_n x_n \psi$$

where ψ is a boolean formula over the variables x_1, \dots, x_n and $Q_1 = \exists$. If θ has k quantifier alternations, partition the sequence Q_1, \dots, Q_n into $k+1$ alternating blocks of identical quantifiers accordingly. For each $i \in [1, k+1]$ we choose numbers $r_i > 0$ such

that the last variable quantified in i th block is x_{r_i} . In this case, we have $r_{k+1} = n$. For each $i \in [1, k]$, we also let Q_i be the quantifier type of the i th block of quantifiers (e.g. $Q_1 = \exists$ by assumption). Let $\bar{v} := (r_1, \dots, r_k)$. Define ϕ to be the \mathcal{L}_{DIV} -formula obtained by replacing each occurrence of the boolean variable x_i in ψ by the atomic proposition $p_i|x$. It is then easy to check that θ is true iff Player \exists has a winning strategy in the input arena (\bar{v}, k, ϕ) . [That is, for each truth evaluation $f : \{x_1, \dots, x_n\} \rightarrow \{0, 1\}$ and each $m \in \mathbb{Z}_{>0}$ satisfying $\mathfrak{G}(m) \in f(x_1) \dots f(x_n)(0+1)^\omega$, it is easy to check that ψ is true under f iff $\langle \mathbb{N}, + \rangle \models \phi(m)$. To complete the proof, observe that the vector \bar{v} is defined in such a way that the quantifiers in θ are faithfully simulated in the buffer game.]

E.4 Proof sketch of Proposition 9.4.4

The proof is similar to the proof of Proposition 9.4.2 with the following modification. We store the buffer values chosen by the Player \exists and \forall in two different counters. At the round r where Player \exists acts, suppose that the current buffer value is n_0 and n_1 . Player \exists rewrites the first counter with a new value n'_0 , while making sure that for each prime number $p \in [1, v_r]$ and any number $j \in [0, p)$ it is the case that $n_0 \equiv j \pmod{p}$ iff $n_1 \equiv j \pmod{p}$. It is not hard to encode this as an EF-formula (with respect to two appropriate OCPs), whose size depends (polynomially) on v_r .

E.5 Proof of Proposition 9.4.5

We show that checking $\text{FO}^4(\text{Reach})$ sentences with equality over $(\mathbb{N}, \text{succ})$ is PSPACE-hard. We shall only make use of the variables x, y, u, v . Since it is easy to construct an OCP generating $(\mathbb{N}, \text{succ})$, it follows that the expression complexity of $\text{FO}^4(\text{Reach})$ over OCPs is PSPACE-hard. To deduce the lower bound for $\text{FO}^4(\text{Reach})$ without equality, observe first that we can define a strict inequality relation in $\text{FO}^4(\text{Reach})$ (without equality) over $(\mathbb{N}, \text{succ})$ as follows:

$$x < y \Leftrightarrow \exists u(\text{succ}(x, u) \wedge \text{Reach}(u, y)).$$

Therefore, the equality relation $x = y$ can simply be expressed as $\neg(x < y \vee y < x)$.

In the following, we shall give a poly-time reduction from QBF. Given a formula

$$\phi = Q_1 x_1 \dots Q_n x_n \psi$$

where each Q_i is either \exists or \forall , Q_1 is \exists , and ψ is a boolean formula over x_1, \dots, x_n , we shall compute a $\text{FO}^4(\text{Reach})$ sentence α over $(\mathbb{N}, \text{succ})$ such that φ is true iff $\langle \mathbb{N}, \text{succ} \rangle \models \alpha$.

To prove this, we shall first show how to succinctly encode the relation

$$\text{DIFF}_i = \{(a, b) \in \mathbb{N} \times \mathbb{N} : |a - b| = 2^i\},$$

for each $i \in \mathbb{N}$, using only four variables. The technique is adapted from Grohe and Schweikardt [GS05]. We show how to define DIFF_i in $\text{FO}^4(\text{Reach})$ by induction on i . For the base case, we shall define $\text{DIFF}_i(x, y)$ and $\text{DIFF}_i(u, v)$ to be, respectively, the formulas $\text{succ}(x, y) \vee \text{succ}(y, x)$ and $\text{succ}(u, v) \vee \text{succ}(v, u)$. For $i > 0$, we define

$$\text{DIFF}_i(x, y) := x \neq y \wedge \exists u \forall v [(v = x \vee v = y) \rightarrow \text{DIFF}_{i-1}(u, v)],$$

which says that there exists a midpoint u in the region $[x, y]$ or $[y, x]$ (depending on whether $x < y$ or $y < x$) whose distance from x and y is 2^{i-1} . Similarly, we can define $\text{DIFF}_i(u, v)$ by interchanging every occurrence of x (resp. y) with u (resp. v). Notice that the size of the formula DIFF_i grows only linearly in i .

From the relation DIFF_i , we can now succinctly define some simple arithmetic on large numbers. Define the relation

$$\text{PLUS}_{2^i}(x, y) \Leftrightarrow x < y \wedge \text{DIFF}_i(x, y),$$

which asserts that $y = x + 2^i$. We shall also define the weak minus relation

$$\begin{aligned} \text{WMIN}_{2^i}(x, y) &= \exists u \text{PLUS}_{2^i}(u, x) \rightarrow \text{PLUS}_{2^i}(y, x) \\ &\quad \wedge \neg \exists u \text{PLUS}_{2^i}(z, x) \rightarrow x = y \end{aligned}$$

which is true if either $y = x - 2^i$ and $y \geq 0$, or $x - 2^i < 0$ and $x = y$. It is easy to see that the size of the formulas PLUS_{2^i} and WMIN_{2^i} grows only linearly in i .

We now construct the formula α encoding the quantified boolean formula φ . In the following, the (meta)variable z_i with odd (resp. even) indices will refer to x (resp. y). We first define a $\text{FO}^4(\text{Reach})$ formula $\beta(z_n)$ such that, for each truth valuation $f : \{x_1, \dots, x_n\} \rightarrow \{0, 1\}$, it is the case that ψ is true under f iff $(\mathbb{N}, \text{succ}) \models \beta(m)$, where m is the number with binary representation $f(x_n)f(x_{n-1}) \dots f(x_1)$. For this, it suffices to show how to succinctly define the relation

$$\text{BIT}_i^j(z_j) \Leftrightarrow a \in [0, 2^j) \text{ and the } i\text{th bit of } z_j \text{ is } 1.$$

for each $j = (i, n]$. This is because if this is the case, then we can obtain $\beta(z_n)$ by simply replacing each occurrence of x_i in ψ by $\text{BIT}_i^n(z_n)$. We first show how to define BIT_i^{i+1} for each $i \in [0, n)$. This relation can simply be defined as

$$\text{BIT}_i^{i+1}(z_{i+1}) := \neg \exists z_i \text{PLUS}_{2^{i+1}}(z_i, z_{i+1}) \wedge \exists z_i (\text{PLUS}_{2^i}(z_i, z_{i+1})).$$

Now for each $j > i + 1$, we may simply define

$$\text{BIT}_i^j(z_j) := \exists z_{j-1} [\text{WMIN}_{2^j}(z_j, z_{j-1}) \wedge \text{BIT}_i^{j-1}(z_{j-1})].$$

Notice that $\text{BIT}_i^n(z_n)$ grows polynomially in n .

Now to define the formula α , we shall define a sequence $\{\alpha_i(z_i)\}_{i=0}^n$ of $\text{FO}^4(\text{Reach})$ formulas with one free variable such that, for each valuation $f : \{x_1, \dots, x_i\} \rightarrow \{0, 1\}$, the formula $Q_{i+1}x_{i+1} \dots Q_n x_n \psi$ is true under f iff $(\mathbb{N}, \text{succ}) \models \alpha_i(m)$, where m is the number with binary representation $f(x_i)f(x_{i-1}) \dots f(x_1)$. [If this sequence is empty, then the number m associated with it is 0.] In this case, we shall set $\alpha := \alpha_0(y) \wedge y = 0$, where $y = 0$ is a shorthand for $\neg \exists x \text{succ}(x, y)$. We construct the formulas $\{\alpha_i(z_i)\}_{i=0}^n$ recursively. We obviously start by setting $\alpha_n(z_n) := \beta(z_n)$. For each $i < n$, assume that we have defined $\alpha_{i+1}(z_{i+1})$. Then, if $Q_i = \exists$, we define

$$\alpha_i(z_i) := \exists z_{i+1} [(\text{PLUS}_{2^i}(z_i, z_{i+1}) \vee z_{i+1} = z_i) \wedge \alpha_{i+1}(z_{i+1})].$$

If $Q_i = \forall$, we define

$$\alpha_i(z_i) := \forall z_{i+1} [(\text{PLUS}_{2^i}(z_i, z_{i+1}) \vee z_{i+1} = z_i) \rightarrow \alpha_{i+1}(z_{i+1})].$$

It is now easy to see that, for each valuation $f : \{x_1, \dots, x_i\} \rightarrow \{0, 1\}$, the formula $Q_{i+1}x_{i+1} \dots Q_n x_n \psi$ is true under f iff $(\mathbb{N}, \text{succ}) \models \alpha_i(m)$, where m is the number with binary representation $f(x_i)f(x_{i-1}) \dots f(x_1)$. This is simply because assigning 1 to x_{i+1} corresponds to adding 2^i to z_i , as reflected in the definition of $\alpha_i(z_i)$.

Finally, it is easy to see that our construction above runs in polynomial time.

E.6 Proof of Proposition 9.4.6

Given each $\phi(x, y) \in \text{FO}^2(\text{Reach})$ and each OCP $\mathcal{P} = (Q, \delta_0, \delta_{>0})$ over actions ACT , we compute in poly-time an EF_S -logic formula ϕ' , an OCP $\mathcal{P}_1 = (Q_1, \delta_0^1, \delta_{>0}^1)$ over ACT_1 , and an OCP $\mathcal{P}_2 = (Q_2, \delta_0^2, \delta_{>0}^2)$ over ACT_2 , and a poly-time computable function

f (resp. f') mapping configurations of \mathcal{P} to configurations of \mathcal{P}_1 (resp. \mathcal{P}_2), such that for any pair of \mathcal{P} -configurations (q, n) and (q', n')

$$\mathfrak{S}_{\mathcal{P}} \models \varphi((q, n), (q', n')) \Leftrightarrow \mathfrak{S}_{\mathcal{P}_1 \times \mathcal{P}_2}, (f(q, n), f(q', n')) \models \varphi'.$$

Therefore, by Theorem 9.4.2, there exists a fixed φ_k such that model checking φ_k over ΠOCP is Σ_k^P -hard.

Suppose that $\text{ACT} = \{\sigma_1, \dots, \sigma_l\}$. Let $\text{ACT}_1 = \{a_1, \dots, a_l\} \cup \{c_1, \text{loop}_1\}$ and $\text{ACT}_2 = \{b_1, \dots, b_l\} \cup \{c_2, \text{loop}_2\}$ be two disjoint alphabets. Define $Q_1 = Q_2 = Q \cup \{g\}$. The purpose of adding a new state g is to define a “global modality”. As we shall see, for every $(q, n), (q', n') \in Q_1 \times \mathbb{N}$ we have $(q, n) \xrightarrow{*}_{\{c_1, \text{loop}_1\}} (q', n')$; a similar statement also holds for \mathcal{P}_2 . We now define the transition functions. The transition functions $\delta_{>0}^1$ (resp. $\delta_{>0}^2$) can be obtained from $\delta_{>0}$ by first renaming each action σ_i with a_i (resp. b_i), and then adding extra transitions $(q, c_1, g, 0)$, $(g, \text{loop}_1, g, +1)$, $(g, \text{loop}_1, g, -1)$, and $(g, c_1, q, 0)$ (resp. $(q, c_2, g, 0)$, $(g, \text{loop}_2, g, +1)$, $(g, \text{loop}_2, g, -1)$, and $(g, c_2, q, 0)$), for each $q \in Q$. Similarly, the transition functions δ_0^1 (resp. δ_0^2) can be obtained from δ_0 by first renaming each action σ_i with a_i (resp. b_i), and then adding extra transitions $(q, c_1, g, 0)$, $(g, \text{loop}_1, g, +1)$, and $(g, c_1, q, 0)$ (resp. $(q, c_2, g, 0)$, $(g, \text{loop}_2, g, +1)$, and $(g, c_2, q, 0)$), for each $q \in Q$. Also, define $\text{ACT}'_1 := \Sigma_1 - \{c_1, \text{loop}_1\}$, $\text{ACT}'_2 := \text{ACT}_2 - \{c_2, \text{loop}_2\}$, $\text{ACT}''_1 := \{c_1, \text{loop}_1\}$, and $\text{ACT}''_2 := \{c_2, \text{loop}_2\}$.

We now define our formula φ' by induction on φ . In particular, we shall construct a function λ mapping formulas of $\text{FO}^2(\text{Reach})$ to EF_S -logic formulas such that, for each $\psi(x, y) \in \text{FO}^2(\text{Reach})$ and $(q, n), (q', n') \in Q \times \mathbb{N}$,

$$\mathfrak{S}_{\mathcal{P}} \models \psi((q, n), (q', n')) \Leftrightarrow \mathfrak{S}_{\mathcal{P}_1 \times \mathcal{P}_2}, ((q, n), (q', n')) \models \lambda(\psi) \quad (*)$$

We then need only set $\varphi' = \lambda(\varphi)$. In the following, recall that the formula $\exists x \psi(x, y)$ can be thought of as $\exists x (\psi(x, y) \wedge x = x)$, and so the variables x and y always occur freely in the formula. First consider the base cases:

- $\psi := \text{Reach}_{(x, y)}$. We set $\lambda(\psi) := \langle R_{\text{ACT}'_1} \rangle (=_{1,2})$.
- $\psi := \text{Reach}_{(y, x)}$. We set $\lambda(\psi) := \langle R_{\text{ACT}'_2} \rangle (=_{1,2})$.
- $\psi := E_{\sigma_i}(x, y)$. We set $\lambda(\psi) := \langle a_i \rangle (=_{1,2})$.
- $\psi := E_{\sigma_i}(y, x)$. We set $\lambda(\psi) := \langle b_i \rangle (=_{1,2})$.

It is easy to check that $(*)$ hold for these. Now consider the inductive cases:

- $\psi := \alpha(x, y) \wedge \beta(x, y)$. We set $\lambda(\psi) := \lambda(\alpha) \wedge \lambda(\beta)$.
- $\psi := \neg\alpha(x, y)$. We set $\lambda(\psi) := \neg\lambda(\alpha)$.
- $\psi := \exists x\alpha(x, y)$. We set $\lambda(\psi) := \langle R_{\text{ACT}_1''} \rangle (\lambda(\alpha) \wedge \neg\langle \text{loop}_1 \rangle)$. We show that (*) holds. Given $(q, n), (q', n') \in \mathcal{Q} \times \mathbb{N}$, we have $\mathfrak{S}_{\mathcal{P}} \models \psi((q, n), (q', n'))$ iff there exists $(q'', n'') \in \mathcal{Q} \times \mathbb{N}$ such that $\mathfrak{S}_{\mathcal{P}} \models \alpha((q'', n''), (q', n'))$ iff (by induction hypothesis) there exists $(q'', n'') \in \mathcal{Q} \times \mathbb{N}$ such that $\mathfrak{S}_{\mathcal{P}_1 \times \mathcal{P}_2}, ((q'', n''), (q', n')) \models \lambda(\alpha)$ iff $\mathfrak{S}_{\mathcal{P}_1 \times \mathcal{P}_2}, ((q, n), (q', n')) \models \lambda(\psi)$ since $\langle R_{\text{ACT}_1''} \rangle$ is a global modality.
- $\psi := \exists y\alpha(x, y)$. We set $\lambda(\psi) := \langle R_{\text{ACT}_2''} \rangle (\lambda(\alpha) \wedge \neg\langle \text{loop}_2 \rangle)$.

Finally, observe that our reduction runs in polynomial time.