

# **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**

This doctoral thesis will present the results of my work into the reanimation of lifeless human tissues.

# Acknowledgements

First paragraph: thank Leonid Libkin and then thank Richard Mayr.

Second paragraph: thank lots of colleagues and people with whom I have had useful discussions: Shunichi Amano, Vince Barany, Pablo Barcelo, Lorenzo Clemente, Claire David, Kousha Etessami, Floris Geerts, Stefan Göller, Julian Guitierrez, Matthew Hague, Jerome Leroux, Christof Löding, Shuai Ma, Christophe Morvan, Filip Murlak, Benjamin Rubinstein, Rahul Santhanam, Tony Tan, Yinghui Wu, Sanming Zhou, ...

Third paragraph: thank family and friends. Thank office mates: Vasilis, Fabricio, Karthik, Christophe Dubach.

# 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

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Specific approaches . . . . .	3
1.2	Generic approaches . . . . .	3
1.3	Contributions . . . . .	3
1.4	Organizations . . . . .	3
<b>2</b>	<b>Preliminaries</b>	<b>4</b>
2.1	General notations . . . . .	4
2.2	Automata theory . . . . .	5
2.2.1	Automata over finite words . . . . .	5
2.2.2	Automata over $\omega$ -words . . . . .	12
2.2.3	Automata over trees . . . . .	13
2.2.4	Automata over infinite trees . . . . .	19
2.2.5	Pushdown automata and context-free grammars . . . . .	20
2.3	Computability and complexity theory . . . . .	21
2.4	Structures and transition systems . . . . .	21
2.5	Logics and properties . . . . .	22
2.5.1	Safety, liveness, and fairness . . . . .	22
2.5.2	Reachability and recurrent reachability . . . . .	23
2.5.3	FO: First-order logic . . . . .	25
2.5.4	FO <sub>REG</sub> (Reach): FO with regular reachability . . . . .	28
2.5.5	HM-logic: Hennessy-Milner Logic . . . . .	29
2.5.6	EF <sub>REG</sub> -logic: HM-logic with regular reachability . . . . .	30
2.5.7	CTL: Computation Tree logic . . . . .	31
2.5.8	LTL: Linear Temporal Logic . . . . .	31
2.5.9	Other logics . . . . .	32

<b>I</b>	<b>Generic Approaches: Algorithmic Metatheorems</b>	<b>33</b>
<b>3</b>	<b>Word/Tree-automatic Systems</b>	<b>34</b>
3.1	Word-automatic systems . . . . .	35
3.1.1	Basic definitions . . . . .	35
3.1.2	Examples . . . . .	37
3.1.3	Basic closure and algorithmic results . . . . .	41
3.1.4	Negative results . . . . .	46
3.2	Tree-automatic systems . . . . .	47
3.2.1	Basic definitions . . . . .	47
3.2.2	Examples . . . . .	49
3.2.3	Basic closure and algorithmic results . . . . .	51
3.2.4	Negative results . . . . .	53
3.3	Other generic frameworks . . . . .	53
3.3.1	Length-preserving word-automatic transition systems . . . . .	53
3.3.2	Presburger transition systems . . . . .	54
3.3.3	Rational transition systems . . . . .	56
3.3.4	$\omega$ -word automatic transition systems . . . . .	57
<b>4</b>	<b>Algorithmic metatheorems for recurrent reachability</b>	<b>58</b>
4.1	The word-automatic case . . . . .	58
4.1.1	Main theorem . . . . .	58
4.1.2	Proof of the main theorem . . . . .	59
4.1.3	A small witness for recurrent reachability . . . . .	66
4.1.4	Two appetizing examples . . . . .	67
4.2	The tree-automatic case . . . . .	69
4.2.1	Main theorem . . . . .	69
4.2.2	Preliminaries . . . . .	70
4.2.3	Proof of the main theorem . . . . .	71
4.2.4	An appetizing example . . . . .	76
4.3	Generalized Büchi condition . . . . .	76
4.4	Recurrent reachability via approximation . . . . .	77
4.5	Connection to Ramseyan quantifiers . . . . .	77
<b>5</b>	<b>Algorithmic metatheorems for logic model checking</b>	<b>78</b>
5.1	Model checking LTL . . . . .	78

5.2	Model checking LTL fragments . . . . .	80
5.3	Model checking $\text{FO}_{\text{REG}}(\text{Reach} + \text{EGF})$ . . . . .	80
5.4	Applications: an appetizer . . . . .	81
<b>6</b>	<b>More applications of algorithmic metatheorems</b>	<b>82</b>
6.1	system 1 etc . . . . .	82
6.2	Pushdown systems with discrete clocks and reversal-bounded counters	82
6.3	Building bigger systems from smaller blocks . . . . .	82
6.4	Experimental results . . . . .	82
<b>7</b>	<b>Limitations of generic approaches</b>	<b>83</b>
7.1	Checking regularity of reachability relations . . . . .	83
7.2	A nonelementary lower bound for HM-logic . . . . .	84
<b>II</b>	<b>Specific Approaches</b>	<b>87</b>
<b>8</b>	<b>Reversal-bounded counter systems with discrete clocks</b>	<b>88</b>
8.1	A Caratheodory-like theorem for linear sets . . . . .	89
8.2	Parikh images of regular languages . . . . .	93
8.2.1	A normal form theorem . . . . .	93
8.2.2	Complementary lower bounds . . . . .	98
8.3	Three simple applications . . . . .	100
8.3.1	Integer programming . . . . .	100
8.3.2	Decision problems for Parikh images of NWAs . . . . .	102
8.3.3	Presburger-constrained graph reachability . . . . .	106
8.4	Applications to model checking . . . . .	109
8.4.1	LTL with complex fairness . . . . .	109
8.4.2	Branching-time logics . . . . .	109
8.5	Other applications . . . . .	111
<b>9</b>	<b>One-counter processes</b>	<b>112</b>
9.1	Application to weak-bisimilarity checking . . . . .	112
<b>10</b>	<b>Networks of one-counter processes</b>	<b>113</b>

<b>III Epilogue</b>	<b>114</b>
<b>11 Conclusions and Future work</b>	<b>115</b>
<b>Bibliography</b>	<b>116</b>
<b>A Proofs from Chapter 4</b>	<b>129</b>
A.1 Proposition 4.1.5 implies necessity in Lemma 4.1.4 . . . . .	129
A.2 Proof of Lemma 4.2.2 . . . . .	130
<b>B Proofs from Chapter 8</b>	<b>132</b>
B.1 Proof of Fact 8.2.3 . . . . .	132
B.2 Proof of Lemma 8.2.5 . . . . .	133
B.3 Proof of Lemma 8.2.6 . . . . .	133
B.4 Proof of Proposition 8.2.10 . . . . .	133
B.5 Proof of Proposition 8.3.2 . . . . .	135



# 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 productivity<sup>1</sup>. 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  $\phi$  in some *logical language* (usually some temporal logic). This reduces the initial problem to checking whether  $\mathcal{S}$  satisfies  $\phi$ , which can then be checked using standard model checking algorithms (e.g. see [CGP99, Sch02]).

---

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

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<sup>+</sup>90, McM93], which is to develop model checking algorithms on *symbolic representations* of the transition system. In the case of [BCM<sup>+</sup>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 values) are used as formal definitions of the intuitive notions of algorithms. Albeit it is the case that model checking infinite-state systems is undecidable in general, 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 naively) 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 semi-algorithms for solving model checking problems over such systems. Although such semi-algorithms are (unsurprisingly) incomplete in general, completeness could sometimes be achieved

over subclasses of the general systems. In contrast, *specific* approaches avoid undecidability by always restricting to non-Turing-powerful formalisms at the outset. We advocate that these two approaches should be viewed as *complemental* instead of *competing against each other* since a disadvantage that is inherent to one approach could sometimes be successfully dealt with using the other approach.

## **1.1 Specific approaches**

## **1.2 Generic approaches**

## **1.3 Contributions**

## **1.4 Organizations**

# Chapter 2

## Preliminaries

In this chapter, we shall fix some mathematical notations that will be used in the sequel, and review basic definitions and results from automata theory, complexity theory, and logic. 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

Let  $\mathbb{N}$  be the set of nonnegative integers. As usual, we use  $\mathbb{R}$ ,  $\mathbb{Q}$  and  $\mathbb{Z}$  to denote, respectively, the set of real numbers, the set of rational numbers, and the set of integers.

We use standard set operations like union ( $\cup$ ), intersection ( $\cap$ ), set difference ( $\setminus$ ), Cartesian product ( $\times$ ), and power set (e.g.  $2^S$  for a given set  $S$ ).

Denote by  $\omega$  the least infinite ordinal.

We use the standard big-oh  $O()$ , big omega  $\Omega()$ , and small-oh  $o()$  notations.

## 2.2 Automata theory

### 2.2.1 Automata over finite words

#### Languages over finite words

An *alphabet*  $\Sigma$  is simply a finite nonempty set of *letters*. We say that  $\Sigma$  is  $k$ -ary if  $|\Sigma| = k$ . A *word* (or *string*) over  $\Sigma$  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  $n = 0$ ,  $w$  is the unique *empty word*  $\epsilon$ . 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  $ababbb$ ). For convenience, we shall often write  $uv$  instead of  $u.v$ . 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  $w^0 = \epsilon$ ,  $w^1 = w$ ,  $w^2 = abab$ ). A word  $w = a_1 \dots a_n$  has *length*  $|w| = n$ . Note that  $|\epsilon| = 0$ . 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  $\Sigma^*$  (resp.  $\Sigma^+$ ) the set of all words (resp. nonempty words) over  $\Sigma$ . In the sequel, when we omit mention of  $\Sigma$  when referring to these notions, we tacitly assume some underlying alphabet  $\Sigma$ .

A *language* (over  $\Sigma$ ) 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  $\Sigma$ , we could also 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

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

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  $\Sigma$  using the standard Backus-Naur Form:

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

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:

- $\mathcal{L}(a) := a$  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  $\Sigma$  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  $\Sigma$ ) are those languages that are generated by some regular expressions (over  $\Sigma$ ). 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  $\Sigma$  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 **States**( $\mathcal{A}$ ) to denote the set  $Q$  of states of  $\mathcal{A}$ . As usual, we shall also treat the transition relation  $\delta$  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*  $\pi$  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 and letters, i.e., each  $p_i \in Q$  and each  $a_i \in \Sigma$ . 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  $\pi$  is a *path on* (input)  $a_1 \dots a_m$ . For convenience, we shall sometimes omit the path labels from  $\pi$ , 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  $\pi$  has length  $|\pi| = m$ , and we let **first**( $\pi$ ) :=  $p_0$  (resp. **last**( $\pi$ ) :=  $p_m$ ) denote the initial (resp. end) state in the path  $\pi$ . 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  $\Sigma$  accepted by  $\Sigma$ . 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 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.

**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 NWA with only one initial state by adding one extra state. In the sequel, we shall often assume that an NWA has only one state  $q_0$  and write  $(\Sigma, Q, \delta, q_0, F)$  instead of  $(\Sigma, Q, \delta, \{q_0\}, F)$ . ■

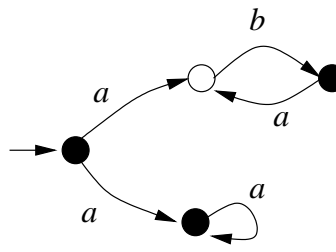


Figure 2.1: An NWA recognizing the language  $\mathcal{L}((ab)^* + a^*)$ .

### 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.  $\|e^*\| := \|e\| + 1$ .

Observe that the number of bits needed to write an expression  $e$  is at most  $O(\|e\| \times \log |\Sigma|)$ .

We now define the complexity measures for NWAs. Given an NWA  $\mathcal{A} = (\Sigma, Q, \delta, Q_0, F)$ , 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  $\delta$ . Then, assuming that each state in  $Q$  occurs in  $\delta$  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  $\Sigma$ , 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  $\Sigma$ :*

- (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:

- $Q := Q^{\mathcal{A}} \cup Q^{\mathcal{B}}$ .
- $Q_0 := Q_0^{\mathcal{A}} \cup Q_0^{\mathcal{B}}$ .
- For each  $a \in \Sigma$ , let

$$\delta(q, a) := \begin{cases} \delta^{\mathcal{A}}(q, a) & \text{if } q \in Q^{\mathcal{A}} \\ \delta^{\mathcal{B}}(q, a) & \text{if } q \in Q^{\mathcal{B}}. \end{cases}$$

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

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  $O(\|\mathcal{A}\| + \|\mathcal{B}\|)$ .

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:

- $Q := Q^{\mathcal{A}} \times Q^{\mathcal{B}}$ .
- $Q_0 := Q_0^{\mathcal{A}} \times Q_0^{\mathcal{B}}$ .
- $\delta((q, q'), a) := (\delta^{\mathcal{A}}(q, a), \delta^{\mathcal{B}}(q', a))$  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  $\|\mathcal{T}\| = \|\mathcal{A}\| \times \|\mathcal{B}\|$ . Furthermore, this construction can be easily implemented in time  $O(\|\mathcal{A}\| \times \|\mathcal{B}\|)$ .  $\square$

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  $\Sigma$  generated *star-free regular expressions* over  $\Sigma$ , which are defined by the following grammars:

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

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  $\Sigma$  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 $\omega$ -words

### Languages over $\omega$ -words

Fix a finite alphabet  $\Sigma$ . An  $\omega$ -word over  $\Sigma$  is a mapping  $w$  from  $\mathbb{Z}_{>0}$  to  $\Sigma$ . As for finite words, we will often think of  $w$  as the infinite sequence  $w(1)w(2)\dots$ . Let  $\Sigma^\omega$  denote the set of all  $\omega$ -words over  $\Sigma$ . 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  $\omega$ -word  $w(i)w(i+1)\dots$ . Given a finite word  $w = a_1 \dots a_n \in \Sigma^*$  and an  $\omega$ -word  $w' \in \Sigma^\omega$ , we define their concatenation as the  $\omega$ -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  $\omega$ -word language over the alphabet  $\Sigma$  is simply a subset of  $\Sigma^\omega$ . As for finite words, we could apply the standard set operations (union, intersection, and complement) to  $\omega$ -word languages. Given a finite word language  $\mathcal{L} \subseteq \Sigma^*$  and an  $\omega$ -word language  $\mathcal{L}' \subseteq \Sigma^\omega$ , we could define their *concatenation* as the  $\omega$ -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.

### $\omega$ -regular languages

As in the case of finite words, we can define the notion of  $\omega$ -regular languages as those  $\omega$ -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  $\omega$ -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  $\pi$  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  $\pi$ . Such an acceptance condition is commonly referred to as *Büchi acceptance condition*. The  $\omega$ -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  $\omega$ -words  $w \in \Sigma^\omega$  that are accepted by  $\mathcal{A}$ . When using NWAs as acceptors of  $\omega$ -word languages, 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  *$\omega$ -regular* (or simply *regular*, when the context is clear) if it is accepted by some NBWA over  $\Sigma$ . For example, the language  $\{a, b\}^* \{a\}^\omega$  is accepted by the NBWA depicted in Figure 2.2.

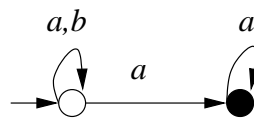


Figure 2.2: An NBWA recognizing the language  $\{a, b\}^* \{a\}^\omega$ .

As in the case of finite words,  $\omega$ -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  $\omega$ -words that capture  $\omega$ -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 the strongly connected component that contains the 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*  $\Upsilon$  is a 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*  $\Upsilon$ , a *tree domain* over  $\Upsilon$  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
- for all  $wi \in D$  and  $1 \leq j < i$ , it is the case that  $wj \in D$ .

For a nonempty set  $\Sigma$  called a *labeling alphabet*, a *tree* over  $\Sigma$  with direction alphabet  $\Upsilon$  is a pair  $T = (D, \tau)$  where  $D$  is a tree domain over  $\Upsilon$  and  $\tau$  is a function mapping  $D$  to  $\Sigma$ . A  $k$ -ary tree over  $\Sigma$  is a tree over  $\Sigma$  with direction alphabet  $\Upsilon = \{1, \dots, k\}$ . Unless

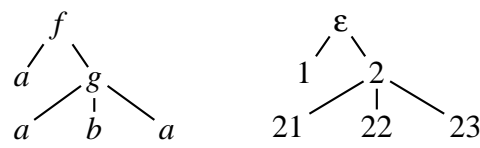


Figure 2.3: An example of a 3-ary tree over  $\{f, g, a, b\}$  (left) depicted together with its tree domain (right) whose elements are ordered by the prefix-of relations.

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  $\Upsilon$  and simply say that  $T$  is a tree over  $\Sigma$  (or just “tree” when  $\Sigma$  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  $\Sigma$ . The elements of  $D$  are also called *nodes*. Therefore, we shall call  $\tau$  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  $i \in \Upsilon$ , we call  $vi$  a *child* of  $v$  and  $v$  the *parent* of  $vi$ . Likewise, if  $vi \in D$  and  $vj \in D$  for some  $i, j \in \Upsilon$ , we say that  $vi$  and  $vj$  are *siblings*. In addition, if  $vw \in D$  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  $\Sigma$ , 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 labeling alphabet  $\Sigma$ , we say that  $T_2$  is a *subtree of  $T_1$  rooted at  $u \in D_1$*  if  $D_2$  coincides with  $\{v : uv \in D_1\}$  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  $\Sigma$  and a node  $u \in D_1$ , we write  $T_1[T_2/u]$  for the tree obtained

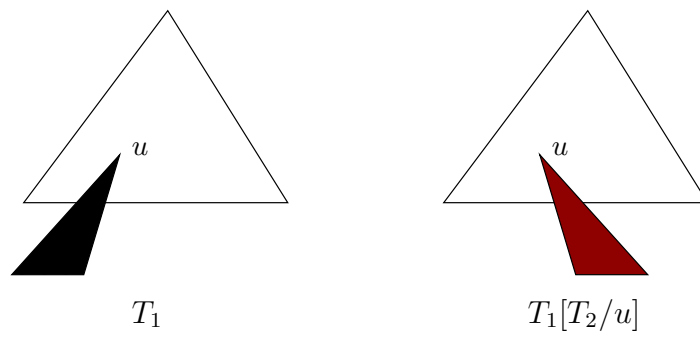


Figure 2.4: A depiction of the subtree substitution operation. Here the subtree rooted at  $u$  on the left is replaced by the tree  $T_2$ , which is the subtree rooted at  $u$  on the right.

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  $T_0, \dots, T_k$  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  $T_0[T_1/u_1, \dots, T_k/u_n]$  to be the tree  $(\dots(T_0[T_1/u_1])\dots)[T_k/u_n]$  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  $\Sigma$  is denoted by  $\text{TREE}_k(\Sigma)$ . A *tree language* over  $\text{TREE}_k(\Sigma)$  is simply a subset of  $\text{TREE}_\Gamma(\Sigma)$ . 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

- $Q$  is a finite set of *states*,
- $Q_0 \subseteq Q$  is a set of *initial* states,

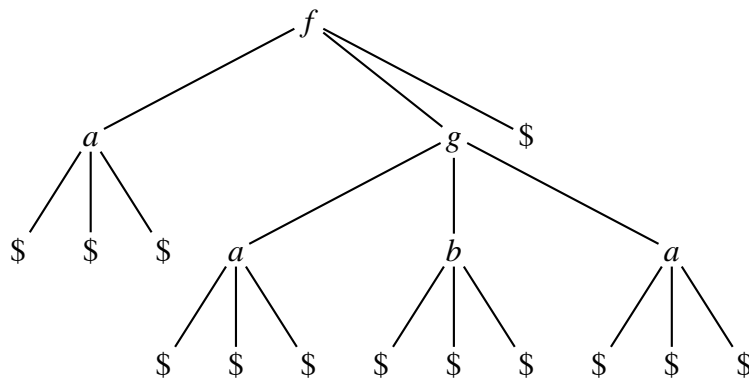


Figure 2.5: The tree depicted here is  $\mathbf{virt}(T)$ , where  $T$  is the tree from Figure 2.3.

- $F \subseteq Q$  is a set of *final* states, and
- $\delta$  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  $\delta$ . 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  $(D', \tau')$  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  $\Sigma$  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  $\delta$  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<sup>+</sup>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 “ $\varepsilon$ -transitions” in our NTAs (which are abundant, among others, in the literature of ground tree rewrite systems and ground tree transducers [CDG<sup>+</sup>07, Löd03, Löd06]), the polynomial-time procedure of removing these  $\varepsilon$ -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<sup>+</sup>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<sup>+</sup>07].

**Proposition 2.2.8** *Checking whether an NTA  $\mathcal{A} = (\Sigma, Q, \delta, Q_0, F)$  over  $k$ -ary trees<sup>1</sup> 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 $\varepsilon$ -transitions

We shall now introduce an extension of NTAs with  $\varepsilon$ -transitions. This is only done for the purpose of convenience when describing the NTAs. Roughly speaking,  $\varepsilon$ -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  $\varepsilon$ -NTA  $\mathcal{A}$  over  $\text{Tree}_k(\Sigma)$  is a tuple  $(\Sigma, Q, \delta, Q_0, F)$ , where  $\Sigma$ ,  $Q$ ,  $Q_0$ , and  $F$  are the same as for NTAs and

- $\delta$  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  $\varepsilon$ -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  $\varepsilon$ -transitions)  $\mathcal{A}' := (\Sigma, Q, \delta', Q_0, F')$  as follows:

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

---

<sup>1</sup>Here,  $k$  is not fixed.

**Proposition 2.2.9** *Given an NTA  $\mathcal{A}$  over  $\text{TREE}_k(\Sigma)$  with  $n$  states, we may construct another 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  $\varepsilon$ -NTA solely for a descriptive purpose.

## 2.2.4 Automata over infinite trees

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

### 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 condition 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  $\omega$ -word automata.

A (*nondeterministic*) *Büchi  $k$ -ary tree automaton*, abbreviated as NBTA, over  $\Sigma$  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  $\rho$  is said to be accepting if, for each infinite path  $\pi = \{v_i\}_{i=0}^\infty$  in the run  $\rho$  (viewed as an infinite  $k$ -ary tree) starting at the root  $\varepsilon$ , there exists infinitely many indices  $i$  such that  $\rho(v_i) \in F$ . In other words, for each infinite path in  $\rho$  starting at the root, the Büchi acceptance condition for  $\omega$ -words is satisfied. The language  $\mathcal{L}(\mathcal{A})$  recognized by  $\mathcal{A}$  is the set of all  $k$ -ary infinite-trees over  $\Sigma$  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  $\Sigma$  is a 4-tuple  $(\Sigma, V, \delta, S_0)$  where

- $V$  is a set of *non-terminals*,
- $S_0 \in V$  is an initial non-terminal,
- $\delta$  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  $\Sigma$  are also often called *terminals*. Given two sequences  $\alpha, \beta \in (V \times \Sigma)^*$  of non-terminals and terminals, we say that  $\beta$  can be *immediately derived* from  $\alpha$ , 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  $\alpha$  can be *derived* from  $\beta$  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  $\Sigma$  and let  $\Sigma_\epsilon := \Sigma \cup \{\epsilon\}$ . A *pushdown automaton (PDA)*  $\mathcal{P}$  over the input alphabet  $\Sigma$  is a tuple  $(\Sigma, \Gamma, Q, \delta, q_0, F)$  where:

- $\Gamma$  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
- $\delta$  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  $\epsilon$ -free if  $\delta$  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 right<sup>2</sup>. A *configuration* of  $\mathcal{P}$  is a pair  $(q, w)$  of state  $q \in Q$  and a *stack content*

---

<sup>2</sup>In 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 purpose.

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

## 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, Sip97] for more details).

Define Turing machines and alternating Turing machines. Define usual complexity classes. Define Minsky's counter machines (see Richard's paper).

## 2.4 Structures and transition systems

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  $\sigma$ -*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** Consider three familiar structures. The first structure is integer with addition  $(\mathbb{N}, +)$ . The addition relation  $+$  is interpreted as a 3-ary relation consisting of

tuples  $(n, m, k) \in \mathbb{N}^3$  such that  $n + m = k$ . The second structure is integer with linear order  $(\mathbb{N}, <)$ , where the 2-ary relation  $<$  consists of pairs  $(n, m) \in \mathbb{N}^2$  such that  $n$  is smaller than  $m$ . The third structure is integer with successor  $(\mathbb{N}, \prec)$ , where the 2-ary relation  $\prec$  consists of pairs  $(n, m) \in \mathbb{N}^2$  with  $m = n + 1$ . ♣

In the sequel, we shall mostly deal with *transition systems*, which are defined as structures over vocabulary  $\sigma$  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 *states*. Furthermore, the edges in each edge relation  $\rightarrow_a$  are also called *actions*. In this case, each name in  $\sigma$  is also said to be an *action label*. Hence, we will also use ACT to denote the vocabulary  $\sigma$ .

**Example 2.4.2** Give an example of transition systems that we care about ♣

## 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.

### 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<sup>+</sup>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  $\rightarrow_{\text{ACT}}$ . 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

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

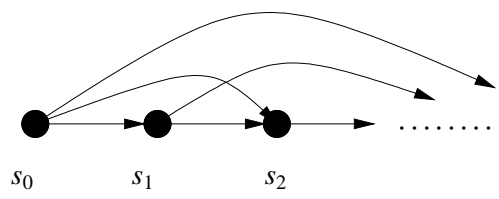


Figure 2.6: An illustration of an infinite sequence witnessing  $s_0 \in \text{Rec}(S')[R]$ . The edges are from the relation  $R$ , while each configuration  $s_i$  (with  $i > 0$ ) must belong to  $S'$ . The configurations in this sequence are not necessarily all different.

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 notation  $\text{Rec}(S')$  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 for some all  $i \in \mathbb{N}$ . The syntax of first-order formulas over the vocabulary  $\sigma$  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  $\phi$  and  $\psi$  are formulas, then so are their conjunction  $\phi \wedge \psi$ , their disjunction  $\phi \vee \psi$ , and the negation  $\neg\phi$ , and (iv) if  $\phi$  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  $\phi$  is said to be *existential positive* if it is of the form  $\exists x_1, \dots, x_n\psi$ , where  $\psi$  is *quantifier-free*, i.e., a boolean combinations of atomic formulas. The *free variables*  $\text{free}(\phi)$  of a first-order formula  $\phi$  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  $\phi$  is a formula with free variables  $x_1, \dots, x_n$ , we write  $\phi(x_1, \dots, x_n)$  to emphasize which free variables the formula  $\phi$  has. A first-order *sentence* is simply a first-order formula  $\phi$  with  $\text{free}(\phi) = \emptyset$ .

Fix a  $\sigma$ -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  $\phi$  is a formula over  $\sigma$ , we may define the notion of *truth* of  $\phi$  in  $\mathfrak{S}$  with respect to  $v$  in the standard way (cf. [Lib04, vD08]). If  $\phi$  is true in  $\mathfrak{S}$  with respect to  $v$ , we also say that  $\mathfrak{S}$  *satisfies*  $\phi$  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  $\phi$ :  $\mathfrak{S} \models \phi[v]$  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  $\sigma$  with the same free variables, we say that  $\phi$  and  $\psi$  are (*logically*) *equivalent*, written  $\phi \equiv \psi$ , if for every  $\sigma$ -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  $\phi$  is defined to be the maximum quantifier nesting depth inside  $\phi$ . 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  $\phi$ , let us push all the negations in  $\phi$  to the atomic level. The *alternation rank*  $AL(\phi)$  of a formula  $\phi$  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  $\phi$ .

### 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  $\sigma$  is of the form

$$v(x_1, \dots, x_k) \leftarrow \phi(y_1, \dots, y_r)$$

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

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

We shall call  $v(x_1, \dots, x_k)$  the *head* of the query  $v$ , while  $\phi(y_1, \dots, y_r)$  is called the *body* of the query  $v$ . 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  $v$  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<sup>k</sup>: 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<sup>k</sup> 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<sup>3</sup> 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.

The reverse interpretation can similarly be done. Therefore, given a first-order sentence  $\varphi$  over the vocabulary consisting of a binary relation name  $<$  and a unary relation name  $U$ , we may define  $\mathcal{L}(\varphi)$  to be the class of finite words  $w$  over  $\{0, 1\}$  such that  $w \models \varphi$ . We say that a language  $\mathcal{L} \subseteq \{0, 1\}^*$  is *definable* in FO (resp. FO<sup>k</sup>) if there exists a formula in FO (resp. FO<sup>k</sup>) such that  $\mathcal{L}(\varphi) = \mathcal{L}$ . 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<sup>3</sup> suffices and the translation to FO<sup>3</sup> 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<sup>3</sup>. Furthermore, the translation from star-free regular expressions to FO<sup>3</sup> 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  $\text{ACT}$  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  $\text{ACT}$ , 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  $\text{ACT}$  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

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

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 formula 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 bisimulations (a.k.a. bisimulation-invariant). Due to its intimate connection with the notion of bisimulation, Hennessy-Milner logic<sup>3</sup> 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  $\text{ACT}$  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  $a \in \text{ACT}$ , 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  $\llbracket \phi \rrbracket_{\mathfrak{S}}$  of an HM-logic formula  $\phi$  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:

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

Given a configuration  $s \in S$ , we also write  $\mathfrak{S}, s \models \phi$  iff  $s \in \llbracket \phi \rrbracket_{\mathfrak{S}}$ .

#### Standard translation to $\text{FO}^2$

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  $\phi$  over  $\text{ACT}$ , there exists an FO formula  $\phi'(x)$  over  $\text{ACT}$  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

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

---

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

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

## Bisimulations

### 2.5.6 $\text{EF}_{\text{REG}}$ -logic: HM-logic with regular reachability

We saw that HM-logic is not capable of expressing reachability. For this reason, we introduce  $\text{EF}_{\text{REG}}$ -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  $\text{EF}_{\text{REG}}$ -logic over the action alphabet  $\text{ACT}$  is the extension of the syntax of HM-logic over  $\text{ACT}$  with the following rule:

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

The semantics of formula of the form  $\text{EF}_{\mathcal{A}}\phi$  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  $\mathfrak{S}, s_0 \models \text{EF}_{\mathcal{A}}\phi$  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  $s_n \in \llbracket \phi \rrbracket_{\mathfrak{S}}$ .

We then define  $\llbracket \text{EF}_{\mathcal{A}}\phi \rrbracket_{\mathfrak{S}}$  to be the set of configurations  $s_0 \in S$  such that  $\mathfrak{S}, s_0 \models \text{EF}_{\mathcal{A}}\phi$ . We define EF-logic to be the restriction of  $\text{EF}_{\text{REG}}$ -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  $\text{EF}_{\Gamma}$  to refer to such a reachability operator, and the shorthand EF to denote  $\text{EF}_{\text{ACT}}$ .

**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  $\text{EF}_{\Gamma}$  does not change the complexity of most 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  $\text{FO}^2$ . In the same manner,  $\text{EF}_{\text{REG}}$ -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<sup>+</sup>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  $\phi$  and  $\psi$  are CTL formulas over ACT, then  $E(\phi \cup \psi)$  is also a CTL formula
- if  $\phi$  is a CTL formula over ACT, then so is  $EG\phi$ .

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:

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

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

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

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

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

## 2.5.8 LTL: Linear Temporal Logic

The syntax of LTL over ACT is

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

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

- $\llbracket a \rrbracket = \{v \in \text{ACT}^\omega : v(1) = a\},$
- $\llbracket \neg\phi \rrbracket = \text{ACT}^\omega - \llbracket \phi \rrbracket,$
- $\llbracket \phi \vee \phi' \rrbracket = \llbracket \phi \rrbracket \cup \llbracket \phi' \rrbracket,$

- $\llbracket \phi \wedge \phi' \rrbracket = \llbracket \phi \rrbracket \cap \llbracket \phi' \rrbracket$ ,
- $\llbracket \mathbf{X}\phi \rrbracket = \{v \in \text{ACT}^\omega : v[1, \infty) \in \llbracket \phi \rrbracket\}$ , 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  $\omega$ -word  $v \in \text{ACT}^\omega$  and an LTL formula  $\phi$  over ACT, we write  $v \models \phi$  iff  $v \in \llbracket \phi \rrbracket$ .

**Proposition 2.5.2 (Vardi-Wolper [VW86a])** *Given an LTL formula  $\phi$  over ACT, we can compute an NBWA  $\mathcal{A}_\phi$  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

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

in  $\mathfrak{S}$ . We define the semantics of LTL over transition systems in the standard way:  $(\mathfrak{S}, v) \models \phi$  iff every  $\omega$ -word  $v \in \text{ACT}^\omega$  realized by  $v$  satisfies  $\phi$ . 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  $\phi$ ). 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  $\phi$ .

## 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 will 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.
- *$\mu$ -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  $\mu$ -calculus.



## **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, BW94, BJNT00, BHV04, DLS02, FL02, KMM<sup>+</sup>97, KMM<sup>+</sup>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. 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 the 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, Bou01, BW94, BJNT00, BHV04, DLS02, KMM<sup>+</sup>97, KMM<sup>+</sup>01, JN00, Nil05, 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 decidability of cer-

tain 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  $\omega$ -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 are those transition systems  $\mathfrak{S} = \langle S, \{\rightarrow_a\}_{a \in \text{ACT}} \rangle$  such that, for some alphabet  $\Sigma$ ,  $S$  is a regular language over  $\Sigma$  and  $\rightarrow_a$  is a “regular relation” over  $\Sigma$ . To define the notion of regular relations, we need a binary operation over  $\Sigma^*$  called *convolution*  $\otimes$ , which computes an encoding of a pair of words over  $\Sigma^*$  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 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  $\left( \begin{bmatrix} 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \end{bmatrix} \right)^* (\{\perp\} \times \{0, 1\})^* \clubsuit$ .

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  $\Sigma^*$ -automatic if  $S$  is a regular language over  $\Sigma$  and each  $\rightarrow_a$  is a regular relation over  $\Sigma$ . It is said to be word-automatic (or just automatic) if it is  $\Sigma^*$ -automatic for some alphabet  $\Sigma$ .

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  $\Sigma^*$ -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  $\mathcal{A}_S$  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  $\mathfrak{S}_\eta$  to denote the transition system of which  $\eta$  is a presentation. A transition system  $\mathfrak{S}'$  over  $\text{ACT}$  is said to be (word-)automatically presentable if it is isomorphic to some automatic system  $\mathfrak{S}$  over  $\text{ACT}$ .

**Remark 3.1.1** 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.

### 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)* (cf. [BEM97, May98, MS85, Tho03]) is simply a PDA without an initial state and the set of final states. 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  $\Sigma_\epsilon$  for which there exists a transition rule in  $\delta$  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  $\Omega_{\perp}^2$  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 [Cau03] are natural generalizations of pushdown systems, where we allow potentially infinitely many rules which are represented using regular languages. More precisely, a *prefix-recognizable system*<sup>1</sup>  $\mathcal{P}$  over ACT is a tuple  $(\text{ACT}, \Gamma, Q, \delta)$  where

- $\Gamma$  is a finite stack alphabet,
- $Q$  is a finite set of states, and
- $\delta$  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  $\Gamma$ .

As for PDSs, a *configuration* is simply a pair  $(q, w) \in Q \times \Gamma^*$  consisting of a state and a word in  $\Gamma^*$ . 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  $\delta$  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  $\delta$  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  $\Gamma^*$  and for some  $\mathcal{A}''$  that accepts all words in  $\Gamma^*$ . A simple example of a transition system that can easily be generated by a prefix-recognizable system (up to isomorphisms) is  $\text{S2S}_{<}$  (see Example 2.4.1). This,

---

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

however, cannot be generated by pushdown systems since the nodes in S2S 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** Minsky’s counter machines [Min67] are well-known Turing-powerful models 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  $\text{ACT}$  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*,
- $\Delta$  is a finite set of *instructions* of the form  $((q, \varphi(X)), a, (q', i_1, \dots, i_k))$ , 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  $(\mathbb{N}, +)$ , 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 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

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

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  $\chi$  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 the time is spent in enumerating the letters in the alphabet  $\Omega$ . 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  $\Omega_{\perp}^2$  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  $\Sigma_{\perp}^n$  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  $\Sigma^*$ -automatic structure over the vocabulary  $\sigma$  is a  $\sigma$ -structure  $\mathfrak{S} = \langle S, \{R_a\}_{a \in \sigma} \rangle$  where  $S$  is a regular language over  $\Sigma$  and  $R_a$  is an  $\text{AR}(a)$ -ary regular relation over  $S$ . A  $\sigma$ -structure is said to be *automatic* if it is  $\Sigma^*$ -automatic for some alphabet  $\Sigma$ .

An *automatic presentation*  $\eta$  of a  $\Sigma^*$ -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  $\Sigma$  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  $\sigma$ -structure is said to be *(word-)automatically presentable* if it is isomorphic to an automatic structure over  $\sigma$ . Remark 3.1.1 for our definition of automatic systems also holds for our definition of automatic structures.

**Example 3.1.6** Presburger arithmetic  $(\mathbb{N}, +)$  is well-known to be  $\{0, 1\}^*$ -automatic (via the reverse binary encoding of numbers; see Example 3.1.4). In fact, the extension of  $(\mathbb{N}, +)$  with the binary relation  $|_2$  is automatic, where for all  $n, m \in \mathbb{N}$ ,

$n \mid_2 m$  iff  $n$  divides  $m$  and  $n = 2^k$  for some  $k \in \mathbb{N}$ . See [BGR10, Blu99, BHMV94] for more details. The structure  $\mathbf{S2S}_{<} = \langle \{0, 1\}^*, \text{succ}_0, \text{succ}_1, \preceq \rangle$  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  $\eta$  of an automatic structure  $\mathfrak{S}_\eta = \langle S, \{R_a\}_{a \in \sigma} \rangle$  over the vocabulary  $\sigma$  and a first-order query  $\mathbf{v}(\bar{x}) \leftarrow \phi(\bar{y})$  over  $\sigma$ , the relation  $[[\mathbf{v}]]_{\mathfrak{S}_\eta}$  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  $\Sigma_\perp^m$ , where  $m = \text{AR}(\mathbf{v})$ , such that for all  $v_1, \dots, v_m \in S$ , it is the case that

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

An NWA for  $[[\mathbf{v}]]_{\mathfrak{S}_\eta}$  can later be obtained by taking a product of  $\mathcal{A}_\phi$  with the regular set  $\mathcal{L}_m := \underbrace{S \otimes \dots \otimes S}_{m \text{ many times}}$ , 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  $\eta$ , 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  $\mathcal{L}(\mathcal{A}_\phi) \cap \mathcal{L}_m = [[\mathbf{v}]]$ . Note that  $\|\mathcal{A}_\phi\| = \|\mathcal{A}_a\|$ . The time taken to construct  $\mathcal{A}_\phi$  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  $\mathcal{L}(\mathcal{A}_{\mathbf{v}'}) \cap \mathcal{L}_m = [[\mathbf{v}']]$  which can be obtained by induction. To obtain an NWA  $\mathcal{A}_\phi$  such that  $\mathcal{L}(\mathcal{A}_\phi) \cap \mathcal{L}_m = [[\mathbf{v}]]$ , we determinize

and then complement  $\mathcal{A}_{v'}$ . The number  $|\mathbf{States}(\mathcal{A}_v)|$  of states of  $\mathcal{A}_v$  is at most exponential in the number  $|\mathbf{States}(\mathcal{A}_{v'})|$  of states of  $\mathcal{A}_{v'}$ . It follows that  $\|\mathcal{A}_v\| \leq 2^{2|\mathbf{States}(\mathcal{A}_{v'})|}$ . The time taken to construct  $\mathcal{A}_v$  is  $2^{O(|\mathbf{States}(\mathcal{A}_{v'})| + m \log(|\Sigma|))}$  on top of the time taken to construct  $\mathcal{A}_{v'}$ .

- Inductive case:  $\varphi := \varphi'(x_{i_1}, \dots, x_{i_s}) \vee \varphi''(x_{j_1}, \dots, x_{j_r})$  for some indices  $1 \leq x_{i_1}, \dots, x_{i_s} \leq m$  and  $1 \leq j_1, \dots, j_r \leq m$ . Define new queries

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

and

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

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

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

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

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

- Inductive case:  $\varphi := \varphi'(x_{i_1}, \dots, x_{i_s}) \wedge \varphi''(x_{j_1}, \dots, x_{j_r})$  for some indices  $1 \leq x_{i_1}, \dots, x_{i_s} \leq m$  and  $1 \leq j_1, \dots, j_r \leq m$ . 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  $\llbracket v \rrbracket$ .

- 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  $\mathcal{A}_{v'}$  such that  $\mathcal{L}(\mathcal{A}_{v'}) \cap \mathcal{L}_{m+1} = \llbracket v' \rrbracket$ , where  $v'$  is defined as

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

Observe that  $\llbracket v \rrbracket = \{(v_1, \dots, v_m) : \exists u \in S((v_1, \dots, v_m, u) \in \llbracket v' \rrbracket)\}$ . Therefore, we need to construct a new NWA  $\mathcal{A}'_{v'}$  from  $\mathcal{A}_{v'}$  in such a way that  $\mathcal{L}(\mathcal{A}'_{v'}) \cap (\mathcal{L}_m \times \Sigma_{\perp}^*) = \llbracket v' \rrbracket$ . More precisely, if  $\mathcal{A}_S = (\Sigma, Q_2, \delta_2, Q_0^2, F_2)$ , 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  $\mathcal{A}_{v'}$  and  $\mathcal{A}'_S$ , we obtain an NWA  $\mathcal{A}'_{v'} = (\Sigma_{\perp}^{m+1}, Q', \delta', Q'_0, F')$

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  $\pi$  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  $\mathcal{A}_{\mathbf{v}} = \|\mathcal{A}_{\mathbf{v}'}\| \times \|\mathcal{A}_S\|$ . The time taken to construct  $\mathcal{A}_{\mathbf{v}}$  is at most  $O(|\Sigma|^{m+1} \times \|\mathcal{A}_{\mathbf{v}'}\| \times \|\mathcal{A}_S\|)$  on top of the time taken to construct  $\mathcal{A}_{\mathbf{v}'}$ .

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  $\mathbf{v}(\bar{x}) \leftarrow \exists y_1, \dots, y_n \phi$ , where  $\phi$  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*

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

*be a conjunctive query over  $\sigma$ , where*

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

*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  $\Sigma^*$ -automatic structure  $\mathfrak{S}_\eta$ , an NWA  $\mathcal{A}_0$  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  $\Sigma^*$ -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 extends to the more general class of existential positive first-order formulas. We shall state it as a proposition, though we do not need it in the sequel.

**Proposition 3.1.3** *Let*

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

*be a first-order formula over the vocabulary  $\sigma$ , where  $\psi$  is a positive boolean combination of atomic propositions with  $h$  conjunctions. Given an automatic presentation*

$\eta$  of an automatic structure  $\mathfrak{S}_\eta$ , an NWA  $\mathcal{A}_\Phi$  accepting  $\llbracket \Phi \rrbracket_{\mathfrak{S}_\eta}$  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 is nonelementary.*

A simple example 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  $\Sigma_1^0$ -complete.*

In fact, the  $\Sigma_1^0$ -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 liveness for automatic transition systems is much harder.

**Proposition 3.1.6** *The recurrent reachability problem for automatic transition systems is  $\Sigma_1^1$ -complete.*

*Proof.* To prove  $\Sigma_1^1$ -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  $\Sigma_1^1$ -complete (e.g. see [Har86, Corollary 6.2]). Therefore, we may use the construction of automatic presentation  $\eta$  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  $\Sigma_1^1$  is to establish a many-to-one reduction to the problem of recurrent state properties for NTMs. Given a presentation  $\eta = \langle \mathcal{A}_S, \{\rightarrow_a$

$\}_{a \in \text{ACT}}\rangle$  over the alphabet  $\Omega$  for the 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  $\Omega$  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  $\mathfrak{S}_\eta$  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  $\mathfrak{S}_\eta$  from the currently remembered configuration  $w$ . If  $w \notin F$ , the NTM  $\mathcal{M}$  will resume the exploration of  $\mathfrak{S}_\eta$  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.  $\square$

## 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  $(D, \tau)$  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

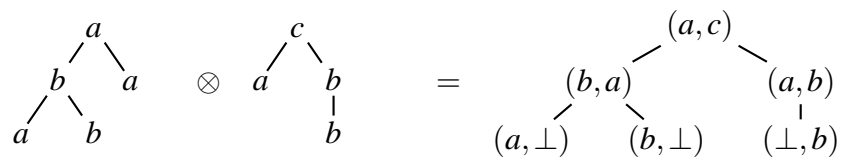


Figure 3.1: A specific example illustrating the convolution operation on binary trees

$R \subseteq \text{TREE}_k(\Sigma) \times \text{TREE}_k(\Sigma)$  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.

**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  $\approx_{\text{dom}}$  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  $\Sigma$ .

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  $\mathfrak{S}_\eta$  to denote the tree-automatic transition system of which  $\eta$  is a presentation. A transition system  $\mathfrak{S}'$  over  $\text{ACT}$  is said to be *tree-automatically presentable* if it is isomorphic to some tree-automatic system  $\mathfrak{S}$  over  $\text{ACT}$ . 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** 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, over automatic systems. ■

### 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,
- $\Sigma$  is a labeling alphabet, and
- $\Delta$  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  $T \in \text{TREE}_k(\Sigma_{\perp}^2)$  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  $\varepsilon$ -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  $\Sigma$  are the same as for GTRS and  $\Delta$  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  $\Sigma$ , we define  $T_1 \otimes \dots \otimes T_n$  to be the  $k$ -ary tree  $T = (D, \tau)$  over the labeling alphabet  $\Sigma_{\perp}^n$ , 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  $\sigma$  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  $\sigma$ -structure is said to be *tree-automatic* if it is  $\text{TREE}_k(\Sigma)$ -automatic for some integer  $k > 0$  and labeling alphabet  $\Sigma$ .

A *presentation*  $\eta$  of a tree-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 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  $\sigma$ -structure is said to be *tree-automatically presentable* if it is isomorphic to a tree-automatic structure over  $\sigma$ . Remark 3.2.1 for our definition of tree-automatic systems also holds for our definition of tree-automatic structures.

**Example 3.2.4** Every  $\Sigma^*$ -automatic structure is easily seen to be 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 prime and

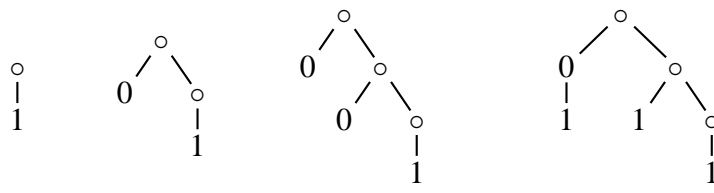


Figure 3.2: Tree representation of the numbers 2,3,5, and 60.

$a_n > 0$ . 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  $\approx_{\text{dom}}$ . Furthermore, it is still tree-automatic when we extend this structure with the binary relations  $\text{succ}_i^a$  ( $1 \leq i \leq k$  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  $\eta$  of a tree-automatic structure  $\mathfrak{S}_\eta = \langle S, \{R_a\}_{a \in \sigma} \rangle$  over the vocabulary  $\sigma$  and a first-order query  $\mathfrak{v}(\bar{x}) \leftarrow \varphi(\bar{y})$  over  $\sigma$ , 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  $\sigma$ , 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  $\mathfrak{S}_\eta$ , an NTA  $\mathcal{A}_\mathfrak{v}$  ac-*

cepting  $\llbracket v \rrbracket_{\mathfrak{S}_\eta}$  of size  $O(\prod_{i=1}^h \|\mathcal{A}_i\|)$  can be computed in time  $O(|\Sigma|^{m+n} \times \prod_{i=1}^h \|\mathcal{A}_i\|)$ .

### 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  $\Sigma_1^0$  and  $\Sigma_1^1$  upper bounds for, respectively, reachability and recurrent reachability also easily carry over to the automatic case.

## 3.3 Other generic frameworks

### 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 alphabet<sup>2</sup>. Length-preserving automatic systems are known to be suitable for modeling *parameterized 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  $\text{post}^*(\mathcal{L}(\mathcal{A}))$  or  $\text{pre}^*(\mathcal{L}(\mathcal{A}))$  for an arbitrary NWA  $\mathcal{A}$ . Note that, unlike in the case of general word-automatic systems, computing  $\text{post}^*(s_0)$  and  $\text{pre}^*(s_0)$ , 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  $\text{post}^*(\mathcal{L}(\mathcal{A}))$  and  $\text{pre}^*(\mathcal{L}(\mathcal{A}))$  need not be regular nor computable in

---

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

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  $w \perp^* \subseteq \text{post}^*(v \perp^*)$  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  $\Sigma$  and a length preserving  $\Sigma^*$ -automatic system  $\mathfrak{S}$ , there exists a configuration  $v$  (or for each configuration  $v$ ) in the language  $\mathcal{L}(\mathcal{A})$  for which it is the case that  $v \in \text{Rec}(\mathcal{L}(\mathcal{A}'))$  is satisfied in  $\mathfrak{S}$ . Since there are only finitely many reachable configurations from any given configuration  $v$ , the infinite path witnessing  $v \in \text{Rec}(\mathcal{L}(\mathcal{A}'))$  in  $\mathfrak{S}$  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 formu-

las  $\phi(x_1, \dots, x_k)$  and  $\psi_a(x_1, \dots, x_{2k})$  over  $(\mathbb{N}, +)$  it is the case that

- $S = \{(i_1, \dots, i_k) \in \mathbb{N}^k : (\mathbb{N}, +) \models \phi(i_1, \dots, i_k)\}$ , 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  $(\mathbb{N}, +) \models \psi_a(i_1, \dots, i_k, j_1, \dots, j_k)$ .

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  $(\mathbb{N}, +)$  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, the word-automatic transition system  $S2S_{<}$  is not Presburger-presentable since the first-order theory of  $S2S_{<}$  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]).

Presburger transition systems (or restrictions thereof) have 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 semi-linear sets have been fully implemented (e.g. see [Boi99, YKB09, YKBB05, hom, 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<sup>+</sup>02, Iba78] and their extensions with one free counter and/or discrete clocks [ISD<sup>+</sup>02, Iba78, DIB<sup>+</sup>00]; and many subclasses of Petri nets [LS04, Esp97, 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; one example is the transition system  $S2S_{<}$ . It turns

out that the framework of Presburger transition systems is powerful enough to capture pushdown systems, although not their reachability relations since S2S can be generated by a fixed pushdown system  $\mathcal{P}$  and the descendant relation  $<$  of S2S by the reachability relation of  $\mathcal{P}$  (see Example 3.1.2).

**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 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 relations 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  $\Sigma$  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  $\epsilon$ . 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 incomparable to the class of tree-automatic transition systems $\star$ . 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.

Not sure  
about  
this  
— An-  
thony

### 3.3.4 $\omega$ -word automatic transition systems

# Chapter 4

## Algorithmic metatheorems for recurrent reachability

This result is from [TL08].

### 4.1 The word-automatic case

#### 4.1.1 Main theorem

**Definition 4.1.1** *A class  $C$  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  $\eta \in C$  over  $\Sigma$ , the transitive closure  $\rightarrow^+$  of  $(\bigcup_{a \in \text{ACT}} \rightarrow_a) \subseteq \Sigma^* \times \Sigma^*$  is regular. Furthermore, the class  $C$  is effectively closed under transitive closure if there exists an algorithm  $\mathcal{M}_C$  that computes an NWA  $\mathcal{R}^+$  recognizing this transitive closure relation for each input presentation  $\eta \in C$ . We say that  $\mathcal{M}_C$  is an effective transitive closure witness (ETC-witness) of  $C$ .*

In the sequel, we shall tacitly assume that  $\mathcal{M}_C$  runs in at least linear time.

**Notation.** Output of  $\mathcal{M}$  on input  $\eta$  is written  $\mathcal{M}(\eta)$  with size  $\|\mathcal{M}(\eta)\| \leq \text{TIME}_{\mathcal{M}}(\|\eta\|)$ .

■

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

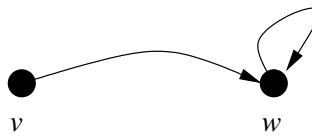


Figure 4.1: A witnessing sequence for  $v \in \text{Rec}_\circ(\mathcal{L}(\mathcal{A}))$  of lasso shape.

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  $O(|v| \times \|\mathcal{A}'\|)$ .

#### 4.1.2 Proof of the main theorem

By closure under transitive closure, we find 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  $\Sigma^*$  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, we have

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

Hence, it suffices to 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. We next show that an NWA  $\mathcal{A}_\circ$  recognizing  $\text{Rec}_\circ(\mathcal{L}(\mathcal{A}))$  can be easily constructed.

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

*Proof.* Observe that, for each word  $v \in \Sigma^*$ ,  $v \in \text{Rec}_\circ(\mathcal{L}(\mathcal{A}))$  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 the construction from the proof of Proposition 3.1.1 on the formula

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

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

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  $i \in \mathbb{N}$ ,  $|\alpha_i| = |\beta_i|$ .

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  $(\bigcup_{a \in \text{ACT}} \rightarrow_a)$ . It is easy to see that the construction runs in time  $O(|\Sigma|^2 \times \|\mathcal{A}\| \times \|\mathcal{R}\| \times (\|\mathcal{R}\| + \|\eta\|))$ .  $\square$

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  $O(|\Sigma|^2 \times \|\mathcal{A}\|^2 \times \|\mathcal{R}\|^3)$ .*

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 exists two sequences  $\{\alpha_i\}_{i \in \mathbb{N}}$  and  $\{\beta_i\}_{i \in \mathbb{N}}$  of words over  $\Sigma$  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  $(\alpha, \beta)$  of  $\omega$ -words over  $\Sigma \cup \{\#\}$ , where  $\#$  is a new symbol not in  $\Sigma$  and

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

The strategy then is to construct an NBWA  $\mathcal{B}$  that accepts  $\omega$ -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 witnessing infinite sequences of the above special form that satisfy further technical restrictions (see below). 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  $\Sigma$  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  $\pi$  of  $\mathcal{A}$  on  $\beta_0\beta_1 \dots$  such that, for all  $i \in \mathbb{N}$ , the NWA  $\mathcal{A}$  accepts  $\alpha_{i+1}$  from  $q$ , where  $q = \pi(|\beta_0 \dots \beta_i|)$ ,*
- (4) *there exists an infinite run  $\pi'$  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  $q' = \pi'(|\beta_0 \dots \beta_{i-1}|)$ .*

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  $\omega$ -chains (say more). Let  $R \subseteq \Sigma^* \times \Sigma^*$  be a (not necessarily transitive) binary relation and  $U \subseteq \Sigma^*$  a language. A (transitive)  $U$ -coloured  $\omega$ -chain in  $R$  from a word  $v \in \Sigma^*$  is an infinite sequence  $\{v_i\}_{i \in \mathbb{N}}$  of *distinct* words in  $\Sigma^*$  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  $(v_i, v_j) \in R$ .

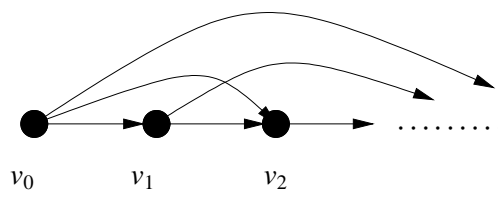


Figure 4.3: An illustration of  $\omega$ -chains.

Figure 4.3 gives an illustration of  $\omega$ -chains. We write  $\text{CHAIN}(U, R)$  to denote the set of words  $v \in \Sigma^*$  from which there exists a  $U$ -coloured  $\omega$ -chain in  $R$ . Observe that  $\text{CHAIN}(\mathcal{L}(\mathcal{A}), \rightarrow^+)$  coincides with  $\text{Rec}_{\rightarrow}(\mathcal{L}(\mathcal{A}))$ .

**Proposition 4.1.5** *Suppose that  $\mathcal{N}$  is an NWA over  $\Sigma$ , 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  $\rho$  of  $\mathcal{N}$  on  $v'v''$ , and a run  $\rho'$  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  $\mathcal{N}$  (resp.  $\mathcal{T}$ ) 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  $\Sigma$  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  $\eta_i$ , 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  $\sigma$ , we may assume that  $|v_i| > |v|$  for all  $i \geq 1$ ; for, otherwise, we may simply omit these words from  $\sigma$ . 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  $\sigma$  with  $i \geq 1$  such that  $\eta$  is not a prefix of  $\eta_i$ . By renaming indices, call the resulting sequence  $\sigma = \{v_i\}_{i \in \mathbb{N}}$  and, for all  $i \geq 1$ , denote by  $\eta_i$  the accepting run of  $\mathcal{N}$  on  $v_i$  that has  $\rho$  as a prefix. Notice that  $\sigma$  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  $\theta_{i_j, i_k}$  (for all  $k > j > 0$ ) has  $u$  as a prefix. Therefore, we can discard all words  $v_i$  ( $i > 0$ ) in  $\sigma$  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  $\pi$  of the NWA  $\mathcal{A}$  and the run  $\pi'$  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  $\pi$  and  $\pi'$  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 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  $\omega$ -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  $\pi$  and  $\pi'$ , 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  $\pi$  and  $\pi'$  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  $\alpha_i$  and  $\beta_i$  are finite.

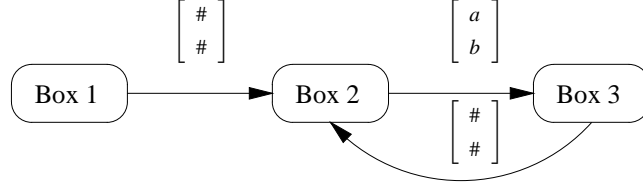


Figure 4.4: A bird's eye view of the Büchi automaton  $\mathcal{B}$

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  $\pi$  and  $\pi'$ . 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

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

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  $\alpha_0$  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  $\Delta$  of  $\mathcal{B}$  to obtain  $\delta^1$ . 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  $\omega$ -words of the form  $\begin{bmatrix} \# \\ \# \end{bmatrix} w$  for some  $\omega$ -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  $\Sigma$  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  $\Sigma$ , we can effectively decide whether  $v_0 \in \text{Rec}(\mathcal{L}(\mathcal{A}))$  and, whenever  $v_0 \in \text{Rec}(\mathcal{L}(\mathcal{A}))$ , 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  $\text{TIME}_{\mathcal{M}}(\|\eta\|) + O(\|\mathcal{M}(\eta)\|^3 \times \|\mathcal{A}\|^2 \times |v_0|)$ .

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  $\omega$ -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  $\{\alpha_i\}_{i \geq 0}$  and  $\{\beta_i\}_{i \in \mathbb{N}}$  of words in  $\Sigma^*$ . 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  $\mathcal{B}'$ . This completes the proof of the Corollary 2.

## 4.1.4 Two appetizing 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 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 NWA 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 (also see ★) or the result of Esparza, Kucera and Schwoon [EKS03] concerning LTL model checking over PDSs with “regular valuations”.

put what  
— An-  
thony

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, EHRS00]).

**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 [EHR00] 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  $q_2v_2 \in \mathcal{L}(\mathcal{A}_{post^*}^C)$ . 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  $\Gamma'$  and each rule in  $\delta'$ , which causes the size of  $\Gamma'$  and  $\delta'$  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 EXPTIME-hard, which can be easily implied from the recent result of Göller [Göl08] on the EXPTIME-completeness of the reachability problem for prefix-recognizable systems.

## 4.2 The tree-automatic case

### 4.2.1 Main theorem

**Definition 4.2.1** *A class  $C$  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  $\eta \in C$ , the transitive closure  $\rightarrow^+$  of  $(\bigcup_{a \in \text{ACT}} \rightarrow_a)$  is tree-regular. Furthermore, the class  $C$  is effectively closed under transitive closure if there exists an algorithm  $\mathcal{M}_C$  that computes an NTA  $\mathcal{R}^+$  recognizing this transitive closure relation for each input presentation  $\eta \in C$ . We say that  $\mathcal{M}_C$  is an effective transitive closure*

witness (ETC-witness) of  $C$ .

**Theorem 4.2.1** *Suppose that  $C$  is a class of tree-automatic presentations that are closed under transitive closure. Then, given a presentation  $\eta \in C$  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  $\text{TREE}_k(\Sigma)$ , the set  $\text{Rec}(\mathcal{L}(\mathcal{A}))$  is tree-regular.*

*Moreover, if  $C$  is effectively closed under transitive closure with an ETC-witness  $\mathcal{M}$ , then an NTA recognizing  $\text{Rec}(\mathcal{L}(\mathcal{A}))$  of size  $O(\dots)$  is computable in time  $\text{TIME}_{\mathcal{M}}(\dots) + \dots$*

Before proving this theorem, we shall define ... (next subsection)

## 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  $\Sigma$  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  $\Sigma$  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  $D' = D \cup \{v_i : v \in D - \{u_1, \dots, u_n\}, 1 \leq i \leq k\}$  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  $\rho$  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  $\Sigma$  is a tree over the labeling alphabet  $\Sigma$  and direction alphabet  $Y = \mathbb{Z}_{\geq 1}$  with a potentially infinite tree domain. Notice that unranked trees 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  $\mathcal{R} = (Q_2, \delta_2, q_0^2, F_2)$ . Without loss of generality, we may assume that  $F_1 = \{q_F^1\}$  and  $F_2 = \{q_F^2\}$  and that there are no transitions from  $q_F^1$  and  $q_F^2$  in the automata  $\mathcal{A}$  and  $\mathcal{R}$ , respectively. 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, it is 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, and 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  $u_i \notin D$ ,
- $D' - \{u_1, \dots, u_n\} \subseteq D$ , and

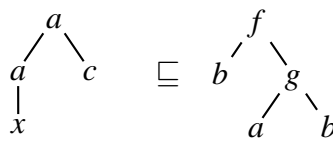


Figure 4.5: An illustration of the relation  $\sqsubseteq$ .

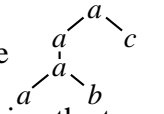
- $u_i = v_i r_i$  for some  $r_i \in \Upsilon$  and  $v_i \in D \cap D'$ .

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, say,  $x_1, \dots, x_n$  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  $\Gamma := \{(t \otimes t'[x_1, \dots, x_r], q, q') : q \in Q_1, q' \in Q_2, r \in \mathbb{N}, t \in \text{TREE}_k(\Sigma), t' \in \text{TREE}_k(\Sigma \cup \{x_1, \dots, x_r\}), \text{ and } t' \sqsubseteq t\}$ , and  $\tau_{\mathfrak{T}}(u) = (\alpha_u \otimes \beta_u[x_1, \dots, x_{r_u}], q_u, q'_u)$  for all  $u \in D_{\mathfrak{T}}$ , such that the following conditions hold:*

1.  $\tau_{\mathfrak{T}}(\varepsilon) = (T \otimes \beta_{\varepsilon}[x_1, \dots, x_{r_{\varepsilon}}], q_0^1, q_0^2)$  for some context tree  $\beta_{\varepsilon}[x_1, \dots, x_{r_{\varepsilon}}]$  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)  $\alpha_u \otimes \beta_u[\alpha_{u_1}, \dots, \alpha_{u_{r_u}}] \in \mathcal{L}(\mathcal{M}^{q'_u})$ ,
- (c) if  $v_1, \dots, v_{r_u}$  are the nodes of  $\beta_u$  labeled by  $x_1, \dots, x_{r_u}$  respectively, then there exist an accepting run  $\rho_u$  of  $\mathcal{A}^{q_u}$  on  $\beta_u[\alpha_{u_1}, \dots, \alpha_{u_{r_u}}]$  and a potentially accepting run  $\rho'_u$  of  $\mathcal{M}^{q'_u}$  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 branches, 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  $\alpha$ 's and  $\beta$ '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  $\sigma$  replaces  $x_k^{u_j}$  by  $\beta_{u_{jk}}[x_1^{u_{jk}}, \dots, x_{r_{u_{jk}}}^{u_{jk}}]$ . Similarly, we define  $T_{i+1}$  to be  $T'[\sigma]$ , where  $\sigma$  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.

**Proposition 4.2.4** *Suppose  $\mathcal{N}$  and  $\mathcal{T}$  are, respectively, an NTA over  $\text{TREE}_k(\Sigma)$  and an NTA over  $\text{TREE}_k(\Sigma_\perp \times \Sigma_\perp)$ . 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  $(T', T') \in \mathcal{L}(\mathcal{T})$ .*

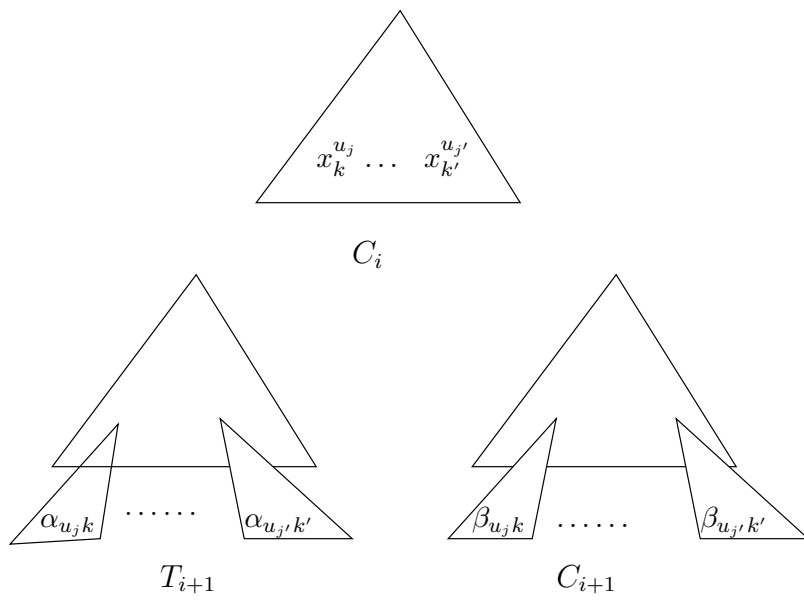


Figure 4.6: An illustration of how the trees  $C_{i+1}$  and  $T_{i+1}$  are obtained from  $C_i$ .

(2) *There exist a  $k$ -ary context tree  $T'[x_1, \dots, x_n] = (D', \tau')$  over the labeling alphabet  $\Sigma$  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  $\sigma$ , 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  $\eta_i$  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  $\sigma$  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  $\eta_i$  is an accepting run  $\mathcal{N}$  on  $T_i$  ( $i \geq 1$ ) such that  $\eta' \preceq \eta_i$ . Notice that  $\sigma$  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  $\sigma$  with  $i \notin V'$  and, again, rename indices. Notice that  $\sigma$  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  $T_i = T'[t_i^1, \dots, t_i^n]$  for some ground trees  $t_i^1, \dots, t_i^n$ . 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  $\sigma_k$ , where  $q_k = \tau_\rho(u_k)$  and  $q'_k = \tau_{\rho'}(u_k)$  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.

#### 4.2.4 An appetizing example

### 4.3 Generalized Büchi condition

We now consider a more general problem 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  $\omega$ -words, one may call this problem *recurrent reachability with generalized Büchi condition*<sup>1</sup>. 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. The next theorem builds on Theorem 4.1.1 and extends it to recurrent reachability with generalized Büchi condition.

**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  $\Sigma$  of an automatic system  $\mathfrak{S}_\eta = \langle S, \{\rightarrow_a\}_{a \in \text{ACT}} \rangle$  and NWA's  $\mathcal{A}_1, \dots, \mathcal{A}_n$  over  $\Sigma$ , 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  $O(\dots)$*

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  $s_0, s_n \in S$

$$s_0 \rightarrow_1 s_n \Leftrightarrow \exists s_1, s_3, \dots, s_{n-1} \in S \left( \bigwedge_{i=0}^{n-1} (s_i \rightarrow^+ s_{i+1}) \wedge \bigwedge_{i=1}^n s_i \in \mathcal{L}(\mathcal{A}_i) \right).$$

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

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

---

<sup>1</sup>Automata over  $\omega$ -words with generalized Büchi conditions, which recognize precisely  $\omega$ -regular languages, are well-studied (e.g. see [Wol00])

The relation  $\rightarrow_1$  is regular by Proposition 3.1.1. Furthermore, an NWA for  $\rightarrow_1$  can be constructed in time

$$O(\|\mathcal{R}\|^n \times \prod_{i=1}^n \|\mathcal{L}(\mathcal{A}_i)\|).$$

Therefore, by Theorem 4.1.1, an NWA for  $Rec(\Sigma^*)[\rightarrow_1]$  of size  $O(\dots)$  can be computed in time  $O(\dots)$ .

## 4.4 Recurrent reachability via approximation

Since the transitive closure  $\rightarrow^+$  of a regular relation  $\rightarrow$  is in general non-recursive (Proposition ??), it makes sense to consider the following approximation problem. 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 in both cases  $R$  may not necessarily be transitive.

## 4.5 Connection to Ramseyan quantifiers

# Chapter 5

## Algorithmic metatheorems for logic model checking

The result in this chapter is from [TL10].

### 5.1 Model checking LTL

Given a finite system  $\mathfrak{F} = \langle Q, \{R_a\}_{a \in \text{ACT}} \rangle$  and an automatic system  $\mathfrak{S}_\eta = \langle S, \{\rightarrow_a\}_{a \in \text{ACT}} \rangle$  presented by an automatic presentation  $\eta$  over  $\Sigma$ , 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 automatic, for which a presentation is computable in time  $O(\|\mathfrak{F}\| \times \|\eta\|)$ .

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

**Theorem 5.1.1** *Suppose that  $C$  is a class of automatic presentations that are effectively closed under transitive closure with an ETC-witness  $\mathcal{M}$  and are closed under product with finite systems. Then, there exists an algorithm which, given an automatic presentation  $\eta$  of a system  $\mathfrak{S}_\eta = \langle S, \{\rightarrow_a\}_{a \in \text{ACT}} \rangle$ , a word  $v_0 \in S$ , and an*

LTl formula  $\phi$ , decides whether  $\mathfrak{S}_\eta, v_0 \models \phi$  in time linear in  $|v_0|$  and polynomial in  $\text{TIME}_{\mathcal{M}}(2^{O(\|\phi\|)} \times \|\eta\|)$ .

*Proof.* Let  $\mathfrak{S}_\eta = \langle S, \{\rightarrow_a\}_{a \in \text{ACT}} \rangle$  be the system presented by the input automatic presentation  $\eta$ . We apply Vardi-Wolper's algorithm (i.e. Theorem 2.5.2) on the negation  $\neg\phi$  of the given LTL formula  $\phi$  yielding an NBWA  $\mathcal{B} = (\text{ACT}, Q, \delta, q_0, F)$  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 product with finite systems to obtain in time  $O(\|\mathcal{B}\| \times \|\eta\|)$  an automatic presentation  $\eta'$  of the system  $\mathcal{B} \times \mathfrak{S}_\eta = \langle QS, \{\rightarrow'_a\}_{a \in \text{ACT}} \rangle$ . Then, for each  $s \in S$ , we have  $(\mathfrak{S}_\eta, s) \models \phi$  iff  $s \notin \llbracket \phi \rrbracket_{\mathfrak{S}}^{\exists}$  iff  $s \notin \text{Rec}(FS, \mathcal{B} \times \mathfrak{S}_\eta)$ . Obviously, an NWA  $\mathcal{N}$  for  $FS$  can be constructed in time  $O(\|\mathcal{B}\| \times \|\eta\|)$ . By Theorem 4.1.1, we may compute an NWA  $\mathcal{A}'$  with  $\mathcal{L}(\mathcal{A}') = \text{Rec}(FS, \mathcal{B} \times \mathfrak{S}_\eta)$  in time  $\text{TIME}_{\mathcal{M}}(\|\eta'\|) + O(\|\mathcal{M}(\eta')\|^3 \times \|\mathcal{N}\|^2)$ , which is polynomial in  $\text{TIME}_{\mathcal{M}}(2^{O(\|\phi\|)} \|\eta\|)$  since  $\|\mathcal{M}(\eta')\| \leq \text{TIME}_{\mathcal{M}}(2^{O(\|\phi\|)} \|\eta\|)$  and  $\|\mathcal{N}\| \leq \|\mathcal{B}\| \times \|\eta\| = 2^{O(\|\phi\|)} \|\eta\|$ . Recall that we tacitly assume that  $\text{TIME}_{\mathcal{M}}$  is at least linear. We then test non-membership of  $v_0$  in  $\mathcal{A}'$  in time  $O(|v_0| \times \|\mathcal{A}'\|)$ , which is linear in  $|v_0|$  and polynomial in  $\text{TIME}_{\mathcal{M}}(2^{O(\|\phi\|)} \|\eta\|)$ . This is in fact also the total running time of the algorithm as one can easily check.  $\square$

**Proposition 5.1.2** *There exists a presentation  $\eta'$  of a fixed automatic system  $\mathfrak{S}_{\eta'} = \langle S, \rightarrow_a, \rightarrow_b \rangle$  and a fixed finite system  $\mathfrak{F}$  over  $\text{ACT} = \{a, b\}$  such that:*

- $\{\eta'\}$  is closed under transitive closure, but
- whenever  $\mathfrak{F} \times \mathfrak{S}_{\eta'} = \langle S', R_a, R_b \rangle$ , the transitive closure of  $R_a \cup R_b$  is not regular (in fact, not even recursive).

*Proof.* Fix a universal Turing machine  $\mathcal{M} = (\Sigma, \Gamma, Q, \delta, q_0, q_F)$  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  $\mathcal{M}$ . 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  $\rightarrow := \rightarrow_a \cup \rightarrow_b = \rightarrow_b$  is also regular. Let  $\mathcal{A}_b$  an NWA that recognizes  $\rightarrow_b$  and define a new automatic presentation  $\eta' = \langle \mathcal{A}_S, \mathcal{A}_a, \mathcal{A}_b \rangle$  which generates the automatic transition system  $\mathfrak{S}_{\eta'} := \langle S, \rightarrow_a, \rightarrow_b \rangle$ . Therefore, the class  $\{\eta'\}$  of automatic presentations is closed under transitive closure.

Now consider the one-state finite system  $\mathfrak{F} = (\{q\}, \rightarrow'_a, \rightarrow'_b)$  over ACT where  $\rightarrow_a := \{(q, q)\}$  and  $\rightarrow_b := \emptyset$ . The transitive closure of the system  $\mathfrak{F} \times \mathfrak{S}_\eta$  coincides with the transitive closure of the system  $\mathfrak{S}_\eta$ , which cannot be recursive (let alone, regular) since the halting problem for  $\mathcal{M}$  is undecidable.  $\square$

## 5.2 Model checking LTL fragments

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

**Theorem 5.2.1** *Suppose that  $\mathcal{C}$  is a class of automatic presentations that are effectively closed under transitive closure with an ETC-witness  $\mathcal{M}$  and are closed under product with finite systems. Then, there exists an algorithm which, given an automatic presentation  $\eta$  of a system  $\mathfrak{S}_\eta = \langle S, \{\rightarrow_a\}_{a \in \text{ACT}} \rangle$ , a word  $v_0 \in S$ , and an  $\text{LTL}_{\text{det}}$  formula  $\varphi$ , decides whether  $\mathfrak{S}_\eta, v_0 \models \varphi$  in time linear in  $|v_0|$  and polynomial in  $\text{TIME}_{\mathcal{M}}(\|\varphi\| \times \|\eta\|)$ .*

**Theorem 5.2.2** *Suppose that  $\mathcal{C}$  is a class of 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 an automatic presentation  $\eta$  of a system  $\mathfrak{S}_\eta = \langle S, \{\rightarrow_a\}_{a \in \text{ACT}} \rangle$ , a word  $v_0 \in S$ , and an  $\text{LTL}_{\text{det}}$  formula  $\varphi$ , decides whether  $\mathfrak{S}_\eta, v_0 \models \varphi$  in time linear in  $|v_0|$ , polynomial in  $\text{TIME}_{\mathcal{M}}(\|\eta\|)$ , and exponential in  $\|\varphi\|$ .*

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

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  $\omega$ -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

- $\mathfrak{S} \models \text{EGF}_{\mathcal{B}}(\phi_1, \dots, \phi_n)(s)$  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  $s_i \models \phi_j$  for infinitely many  $i \in \mathbb{N}$ .

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  $\Gamma \subseteq \text{ACT}$ .

**Theorem 5.3.1** *Supposed that  $\mathcal{C}$  is a class of automatic systems that are effectively closed under transitive closure and closed under product with finite systems. Then, given a presentation  $\eta \in \mathcal{C}$  over  $\Sigma$  of an automatic system  $\mathfrak{S}_\eta = \langle S, \{\rightarrow_a\}_{a \in \text{ACT}} \rangle$  and an  $\text{FO}_{\text{REG}}(\text{Reach} + \text{EGF})$  formula  $\phi(\bar{x})$ , the set  $\llbracket \phi \rrbracket$  is effectively regular. That is, the  $\text{FO}_{\text{REG}}(\text{Reach} + \text{EGF})$  theory is decidable over  $\mathcal{C}$ .*

## 5.4 Applications: an appetizer

# **Chapter 6**

## **More applications of algorithmic metatheorems**

**6.1 system 1 etc**

**6.2 Pushdown systems with discrete clocks and reversal-bounded counters**

**6.3 Building bigger systems from smaller blocks**

Use Wöhrle & Thomas.

**6.4 Experimental results**

# Chapter 7

## Limitations of generic approaches

In this chapter we show that model checking automatic transition systems is non-elementary even using modal logic.

### 7.1 Checking regularity of reachability relations

Put  $\Pi_2^0$ -completeness proof of the problem  $P$  of checking whether a *given* binary regular relation  $R$  is a transitive closure of a binary regular relation  $R'$ . After this, put a proof that checking regularity of a binary relation is in  $\Sigma_3^0$  but not in  $\Pi_3^0$ . Proof is as follows. If it were in  $\Pi_3^0$ , then it's in  $\Delta_3^0$  and so is Turing reducible to  $P$ . Then, assuming this, we can show that the  $\Sigma_3^0$ -complete problem of checking regularity of language of a Turing machine is recursive in the  $\Pi_2^0$ -complete problem  $P$  (i.e. it's in  $\Delta_3^0$ ), which contradicts the fact that arithmetic hierarchy is strict. To solve regularity of the language of a Turing machine, we are given a Turing machine  $T$  and assume that on an input word  $w$ , the machine starts in  $q_0w$  and uses the space to the right of  $w$  as its workspace without ever erasing  $w$ . We also assume that the machine accepts at configuration  $q_Fw$ . (ok wrong argument)

**Theorem 7.1.1** *Given a presentation  $\eta$  over the alphabet  $\Sigma$  of a word/tree automatic system  $\mathfrak{G}_\eta = \langle S, \{\rightarrow_a\}_{a \in ACT} \rangle$  and an NWA  $\mathcal{A}_R$  over  $\Sigma_\perp \times \Sigma_\perp$  recognizing a relation  $R \subseteq \Sigma^* \times \Sigma^*$ , checking whether  $R$  is the transitive closure of  $(\bigcup_{a \in \Sigma} \rightarrow_a)$  is  $\Pi_0^2$ -complete.*

## 7.2 A nonelementary lower bound for HM-logic

**Theorem 7.2.1** *There exists a fixed automatic transition system  $\mathfrak{S} = \langle S, \{\rightarrow_a\}_{a \in \text{ACT}} \rangle$  and a state  $s_0 \in S$  such that checking whether  $(\mathfrak{S}, s_0)$  satisfies a given HM-logic formula is nonelementary.*

We shall now give a proof for this theorem. Recall that  $\text{S2S}_< := (\{0, 1\}^*, \prec_0, \prec_1, <)$  is the infinite binary tree with a descendant relation, i.e.,  $\prec_0 := \{(w, w0) : w \in \{0, 1\}^*\}$ ,  $\prec_1 := \{(w, w1) : w \in \{0, 1\}^*\}$ , and  $< := \{(w, wv) : w, v \in \{0, 1\}^*\}$ . We shall start with an observation that the  $\text{FO}^4$  theory of  $\text{S2S}_<$  is nonelementary.

**Proposition 7.2.2** *The  $\text{FO}^4$  theory of  $\text{S2S}_<$  is nonelementary.*

Although the first-order theory of  $\text{S2S}_<$  was proved to be nonelementary in [CH90], it is not easy to see whether  $\text{FO}^k$  suffices from the proof. Nevertheless, one can easily show that  $\text{FO}^4$  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<sup>1</sup> over the alphabet  $\{0, 1\}$  is nonelementary.

**Proposition 7.2.3** *The  $\text{FO}^3$  theory of the class  $\mathcal{C}$  of finite linear orders with a unary predicate is nonelementary.*

Observe that each structure in  $\mathcal{C}$  can be thought of as a binary word and vice versa. 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  $U = \{2, 3\}$ . Therefore, for a first-order sentence  $\phi$  over  $\mathcal{C}$ , the set  $\llbracket \phi \rrbracket$  of binary words that satisfy  $\phi$  can be thought of as a language over  $\Sigma$ . The gist of the proof of Proposition 7.2.3 is that a given star-free regular expression  $e$  can be converted in polynomial time into an equivalent first-order sentence  $\phi_e$  over  $\mathcal{C}$ , as was observed in [EVW02]. Therefore, checking whether  $\mathcal{L}(e) = \mathcal{L}(e')$  for two given star-free regular expressions  $e$  and  $e'$  can be reduced to checking whether the first-order sentence  $\phi_e \leftrightarrow \phi_{e'}$  is true in  $\mathcal{C}$ . The desired nonelementary lower bound in Proposition 7.2.3 can then be deduced by using [Sto74]. To deduce Proposition 7.2.2, we may simply use the polynomial time reduction in [CH90] which, given a first-order sentence  $\phi$  over  $\mathcal{C}$ , outputs a first-order sentence  $\psi$  with one

---

<sup>1</sup>Star-free regular expressions are regular expressions without Kleene star, but instead with complementation operator (e.g. see [Lib04])

extra variable such that  $\phi$  is true in  $\mathcal{C}$  iff  $(\text{S2S}_{<, \varepsilon}) \models \psi$ . By Proposition 7.2.3, the proof for Proposition 7.2.2 is complete.

Now define the transition system

$$\begin{aligned} \mathfrak{T} := & \langle \{0, 1\}^* \times \{0, 1\}^* \times \{0, 1\}^* \times \{0, 1\}^*; \\ & \{\prec_0^i\}_{i=1}^4, \{\prec_1^i\}_{i=1}^4, \{\prec_i\}_{i=1}^4, \{=_{i,j}\}_{1 \leq i < j \leq 4}, \{G_i\}_{i=1}^4 \rangle. \end{aligned}$$

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.
- $\prec_i := \{(\overline{w}, \overline{w}') : w_i \prec w'_i \text{ and } \forall j \neq i (w_j = w'_j)\}$ . 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  $\mathfrak{T}$ .

**Lemma 7.2.4** *The transition  $\mathfrak{T}$  is automatically presentable.*

*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  $\Sigma$  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  $\mathfrak{T}$ .  $\square$

The following lemma is now sufficient to deduce Theorem 7.2.1.

**Lemma 7.2.5** *Checking whether a given HM-logic formula  $\varphi$  over  $\mathfrak{T}$  is satisfied by  $(\mathfrak{T}, (\varepsilon, \varepsilon, \varepsilon, \varepsilon))$  is nonelementary.*

*Proof.* We shall give a polynomial time reduction from the  $\text{FO}^4$  theory of  $\text{S2S}_{<}$ . More precisely, we give a polynomial time computable function  $\lambda$  from  $\text{FO}^4$  formulas  $\varphi(x_1, \dots, x_4)$  to HM-logic formulas  $\lambda(\varphi)$  over the vocabulary of  $\mathfrak{T}$  such that, for each  $v_1, \dots, v_4 \in \{0, 1\}^*$ ,

$$\text{S2S}_{<} \models \varphi(v_1, \dots, v_4) \iff \mathfrak{T}, (v_1, \dots, v_4) \models \lambda(\varphi) \quad (*)$$

The function  $\lambda$  is defined by induction on  $\text{FO}^4$  formulas over  $\text{S2S}_{<}$ . First consider the three base cases:

- We set  $\lambda(x_i \prec_0 x_j) := \langle \prec_0^i \rangle \langle =_{i,j} \rangle \top$ .
- We set  $\lambda(x_i \prec_1 x_j) := \langle \prec_1^i \rangle \langle =_{i,j} \rangle \top$ .
- 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  $\lambda(\varphi \wedge \varphi') := \lambda(\varphi) \wedge \lambda(\varphi')$ . It is easy to see that (\*) holds by inductive hypothesis.
- We set  $\lambda(\neg \varphi) := \neg \lambda(\varphi)$ . 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  $\text{S2S}_{<} \models \psi(v_1, \dots, v_4)$  iff there exists  $v'_1 \in \{0, 1\}^*$  such that  $\text{S2S}_{<} \models \varphi(v'_1, v_2, v_3, v_4)$ . 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.

$\square$

Say that  $\varphi(x_1, \dots, x_4)$  doesn't mean that all these  $x_i$ 's are free — Anthony

## **Part II**

### **Specific Approaches**

# Chapter 8

## Reversal-bounded counter systems with discrete clocks

Minsky counter machines are well-known models of computation. They are Turing complete. Therefore, restriction is needed. Here we consider reversal-bounded counter systems. Extension also considered: with discrete clocks.

Parikh's Theorem [Par66], a celebrated theorem in automata theory, states that the sets of letter-counts (a.k.a. Parikh images) of regular languages and context-free languages are effectively semilinear. Several proofs for this result with different flavours and techniques exist in the literature (e.g. see [Esp97, Koz97, SSMH04, VSS05]). On the other hand, 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. In particular, although the algorithms proposed in [SSMH04, VSS05] produce existential Presburger formulas in polynomial-time from a given NWA or CFG, the number of quantifiers and the number of conjunctions produce depend also on the number of states or the number of terminals/nonterminals in the input (i.e. not only on the size of the alphabet). A much improved construction has been discovered independently in [Kop10] and [To10] (see also the merged paper [KT10]). In particular, this result subsumes Chrobak-Martinez's Theorem<sup>1</sup> [Chr86, Mar02], which proves this in the case of unary alphabet (say more ...). In this section, we will give an improved construction of the Parikh's Theorem using the construction from [To10].

---

<sup>1</sup>Unfortunately, their proofs contain a subtle error, which were only recently fixed in [To09b]



## 8.1 A Caratheodory-like theorem for linear sets

**Theorem 8.1.1** *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  $\mu = O(m^{2k}(k^2 a)^{2k^2+3k})$ . 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]. Proof is given in the appendix.

**Lemma 8.1.2** *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 8.1.1. Intuitively, Lemma 8.1.2 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 much 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 8.1.1. First invoke Lemma 8.1.2 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  $\preceq_j$  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  $\mathbf{v} \preceq_j \mathbf{w}$  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  $\preceq_j$  is a well-founded partial order, and characterizes  $\preceq_j$ -minimal elements.

**Lemma 8.1.3** *The relation  $\preceq_j$  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  $\preceq_j$ -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  $\preceq_j$  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  $\trianglelefteq_j$  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,  $\trianglelefteq_j$  is a well-founded partial order on  $\mathbf{cone}_{\mathbb{N}}(V) \cap \mathbf{cone}(S_j)$ .

Given a  $\trianglelefteq_j$ -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  $\trianglelefteq_j$ -minimal, we could find another vector  $\mathbf{v}' \in \mathbf{cone}_{\mathbb{N}}(V) \cap \mathbf{cone}(S_j)$  such that  $\mathbf{v}' \trianglelefteq_j \mathbf{v}$ . 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 8.1.3 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  $\trianglelefteq_j$ -minimal vectors  $\mathbf{v}$ .

**Lemma 8.1.4** *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  $\trianglelefteq_j$ -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 8.1.3, there exists a  $\trianglelefteq_j$ -minimal vector  $\mathbf{v}'$  satisfying  $\mathbf{v}' \trianglelefteq_j \mathbf{v}$ . Observation (O3) implies that  $\mathbf{v} \in P(\mathbf{v}'; S_j)$ .  $\square$

Note also that if  $V \subseteq \mathbb{N}^k$ , then all  $\trianglelefteq_j$ -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  $\trianglelefteq_j$ -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  $\trianglelefteq_j$ -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  $\mathbf{A}$  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 8.1.5** *For every  $\preceq_j$ -minimal vector  $\mathbf{v} \in \mathbf{cone}_{\mathbb{N}}(V) \cap \mathbf{cone}(S_j)$ , let  $\mathbf{v}_0 := [\mathbf{v}]$  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  $A\mathbf{x} = \mathbf{v}_0$  ( $\mathbf{x} \succeq \mathbf{0}$ ).*

*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  $A\mathbf{u}^T = \mathbf{v}_0$ . 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  $\sum_{i=1}^m x_i \mathbf{v}_i = \mathbf{v}$  ( $\mathbf{v} \succeq \mathbf{0}$ ) 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  $A\mathbf{x} = \mathbf{v}_0$ , for some  $c_1, \dots, c_m \in \mathbb{N}$ . We shall see now that the maximum absolute value  $B$  of numbers appearing in  $U$  exist, 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 8.1.6** *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  $A\mathbf{x} = \mathbf{b}$  ( $\mathbf{x} \succeq \mathbf{0}$ ) 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 8.1.1 and gives us the desired bounds for the parameter  $\mu$  and the maximum absolute value of entries of each  $w_i$  in Theorem 8.1.1. 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 8.1.3. All in all, this could be implemented to run in time  $2^{O(k \log m + k^2 \log(ka))}$ .

## 8.2 Parikh images of regular languages

### 8.2.1 A normal form theorem

In this section, we shall apply Theorem 8.1.1 to obtain a normal form theorem for Parikh images of NWAs.

**Theorem 8.2.1** *Let  $\mathcal{A}$  be an NWA with  $n$  states over an alphabet  $\Sigma$  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*

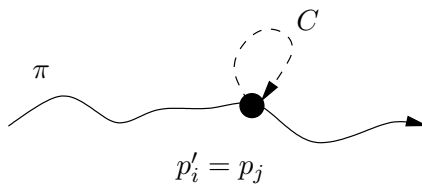


Figure 8.1: The path  $\pi$  (solid line) meets with the cycle  $C$  (broken line) at the state  $p'_i = p_j$  (filled circle).

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  $\pi$  of  $\mathcal{A}$  on  $w$  from some (not necessarily initial) state  $p$  to itself. The cycle  $\pi$  is said to *witness*  $\mathbf{v}$ . Observe that the sum of the components of any cycle type cannot exceed  $n$ .

### Characterization of $\mathcal{P}(L(A))$

We start with a characterization of the Parikh image of  $A$  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  $\pi$  *meet* at state  $p'_i = p_j$ ; see Figure 8.1. Now define  $T_\pi$  to be the linear set  $P(\mathcal{P}(\pi); S_\pi)$ .

**Lemma 8.2.2** *The following identity holds:*

$$\mathcal{P}(\mathcal{L}(\mathcal{A})) = \bigcup_{\pi} T_\pi,$$

where  $\pi$  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

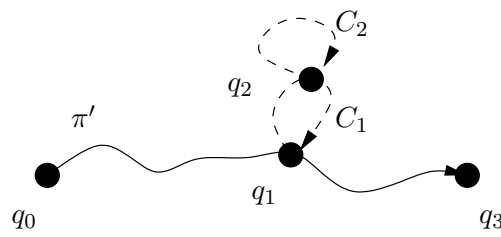


Figure 8.2: The path  $\pi$  is defined as follows: from  $q_0$  it walks to  $q_1$ , continues to  $q_2$  (via half of cycle  $C_1$ ), takes the cycle  $C_2$  once, and then proceeds to  $q_1$  (via the rest of cycle  $C_1$ ) and straight to  $q_3$ . The decomposition is  $\pi'$  (in solid line) and 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 8.2.3** *Given an NWA  $\mathcal{A}$  with  $n$  states and a path  $\pi$  in  $\mathcal{A}$  from  $q$  to  $q'$ , there exist a simple path  $\pi'$  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 8.2.

*Proof of Lemma 8.2.2.* ( $\subseteq$ ) Assume that  $\mathbf{v} \in \mathcal{P}(\mathcal{L}(\mathcal{A}))$  and  $\sigma = p_0 a_1 p_1 \dots a_r p_r$  be an accepting run in  $\mathcal{A}$  such that  $\mathcal{P}(\sigma) = \mathbf{v}$ . We shall construct another accepting run  $\sigma'$  in  $\mathcal{A}$  of length at most  $(n-1)^2$ . For each state  $q$  occurring in  $\sigma$ , 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 8.2.3, each subpath  $\sigma[i_j, i_{j+1}]$  of  $\sigma$  can be decomposed into a simple path  $\pi_j$  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)  $\pi_j$ . For example, in Figure 8.2 the cycle  $C_2$  does not meet with the path  $\pi'$ . 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  $\sigma$ . Thus, we simply define  $\sigma'$  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  $\pi$  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  $\pi$  and satisfies  $\mathcal{P}(C_i) = \mathbf{v}_i$ . We can construct an accepting path  $\sigma$  in  $\mathcal{A}$  with  $\mathcal{P}(\sigma) = \mathbf{v}$  as follows: start from  $\pi$  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  $\pi$ .  $\square$

As an immediate corollary of Lemma 8.2.2, we have:

**Proposition 8.2.4** *Let  $\mathcal{A}$  be an NWA with  $n$  states over an alphabet  $\Sigma$  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 8.2.10 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 8.2.1 is essentially immediate from Proposition 8.2.4 and Theorem 8.1.1. 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  $\pi$  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  $\delta$  of  $\mathcal{A}$ .

**Lemma 8.2.5** *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  $\pi$  with  $\mathcal{P}(\pi) = \mathbf{v}$  can be *uniquely* decomposed into three consecutive path segments  $\pi_1$ ,  $\pi_2$ , and  $\pi_3$  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  $\pi_2$  contains the *first* occurrence of the letter  $a_i$  in the path  $\pi$ . 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 8.2.5 and dynamic programming.

**Lemma 8.2.6** *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  $\Gamma_i$  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  $\Gamma_i$  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 8.2.2, Theorem 8.1.1, 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  $\pi$  is taken over all paths in  $\mathcal{A}$  of length  $i$  from  $q_0$  to  $q_j$ . By Lemma 8.2.2, 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 8.2.7** *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  $\pi$  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  $\Gamma_j$  can be “used” since  $q_j$  is visited. The proof is in the appendix.

To finish the proof of Theorem 8.2.1, 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 8.1.1 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: duplicate 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 8.2.7.

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 8.1.1. 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 8.1.1. All in all, our algorithm runs in time  $2^{O(k^2 \log(nk))}$ , which is also the complexity of the entire procedure.

## 8.2.2 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 8.2.8** *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 8.2.1. In addition, observing that the DWAs that we constructed have equivalent regular expressions of size  $O(n)$ , Proposition 8.2.8 also gives lower bounds for Parikh images of regular expressions.

Next, we show that Theorem 8.2.1 *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 8.2.9** *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  $\Sigma_n$  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 [PSwW02].

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  $\varepsilon$  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 8.2.4, even when restricted to DWAs.

**Proposition 8.2.10** *For each positive integer  $n > 2$ , there exists a DWA  $\mathcal{A}_n$  with  $2n + 3$  states such that if  $\mathcal{P}(\mathcal{L}(\mathcal{A}_n))$  can be represented as a union of the linear sets  $P(\mathbf{v}_1; S_1), \dots, P(\mathbf{v}_r; S_r)$ , 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  $A_n$  has an equivalent regular expression of size  $O(n)$  as well, therefore yielding the same quadratic lower bound for regular expressions.

## 8.3 Three simple applications

In this section, we shall give three simple applications of our main results in previous sections: (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. Nevertheless, these applications are indeed quite general and have relevance beyond model checking. We put other applications of our main results in Section 8.5.

### 8.3.1 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, 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  $A\mathbf{x} = \mathbf{b}$  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

$$A = \left[ \begin{array}{c|c} A_1 & A_2 \\ \hline A_3 & 0 \end{array} \right] \quad (8.1)$$

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 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 8.3.1** *For fixed integers  $k > 0$  and  $m > 0$ , the problem  $\text{IP}_{k,m}$  is solvable in polynomial-time.*

*Proof.* Let  $A\mathbf{x} = \mathbf{b}$  be the given integer program, where  $A$  is of the form given in Equation 8.1 above. Let  $m_1 = m' - m$ . Write the given integer program  $A\mathbf{x} = \mathbf{b}$  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 8.1.1, 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

$$\text{cone}_{\mathbb{N}}(V) = \bigcup_{i=1}^r P(\mathbf{w}_i; S_i),$$

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  $|S_i| = \text{rank}(V) \leq k$ , and  $r = O(m_1^{2k}(k^2a)^{2k^2+3k})$ . Let  $d = \text{rank}(V)$ ,

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:

$$\begin{array}{cccccccc}
a_{1,1}x_1 & + & \dots & + & a_{1,m}x_m & + & c_{1,i_1}y_1 & + & \dots & + & c_{1,i_d}y_d & = & b_1 - s_1 \\
\vdots & & \ddots & & \vdots & & \vdots & & \ddots & & \vdots & & \vdots \\
a_{k,1}x_1 & + & \dots & + & a_{k,m}x_m & + & c_{k,i_1}y_1 & + & \dots & + & c_{k,i_d}y_d & = & b_k - s_k \\
a_{k+1,1}x_1 & + & \dots & + & a_{k+1,m}x_m & & & & & & & = & b_{k+1} \\
\vdots & & \ddots & & \vdots & & & & & & & & \vdots \\
a_{k',1}x_1 & + & \dots & + & a_{k',m}x_m & & & & & & & = & b_{k'}.
\end{array}$$

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.  $\square$

### 8.3.2 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], though we intend to give different proofs 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 [Esp97, Huy83, VSS05]). It turns out that this upper bound is tight.

**Proposition 8.3.2** *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 8.3.2, the membership problem for Parikh images of NWAs becomes solvable in polynomial time when the size  $k$  of the alphabet is fixed.

**Proposition 8.3.3 ([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 8.3.1. That is, we first use Theorem 8.2.1 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 for (as in Proposition 8.3.1).

Let us finally remark that Proposition 8.3.3 cannot be extended to CFGs (or equivalently PDAs).

**Proposition 8.3.4 ([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 [Esp97, 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 8.2.9. In fact, the lower bound is optimal since the membership problem of CFGs is solvable in NP [Esp97, 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  $m_1, \dots, m_k \in \mathbb{N}$ ,

$$(\mathbb{N}, +) \models \varphi_i(m_1, \dots, m_k) \Leftrightarrow (m_1, \dots, m_k) \in \mathcal{P}(\mathcal{L}(G_i)).$$

Testing whether  $\mathcal{P}(\mathcal{L}(G_1)) \cap \mathcal{P}(\mathcal{L}(G_2)) \neq \emptyset$  then corresponds to checking whether

$$(\mathbb{N}, +) \models \exists x_1, \dots, x_k (\varphi_1(x_1, \dots, x_k) \wedge \varphi_2(x_1, \dots, x_k)),$$

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 8.2.1 holds for NWAs with negative inputs.

**Proposition 8.3.5** *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  $\Sigma$  with (respectively)  $n_1$  and  $n_2$  states, we may apply Theorem 8.2.1 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 8.3.6** *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  $\psi$  is quantifier-free and, for each sequence  $i_1, \dots, i_k$  of nonnegative integers, it is the case that

$$(\mathbb{N}, +) \models \varphi(i_1, \dots, i_k) \Leftrightarrow (i_1, \dots, i_k) \in P(\mathcal{B}).$$

Furthermore,  $\|\varphi\|$  is linear in  $\|\mathcal{B}\|$  (even with binary representation of numbers) and that  $\varphi$  can be computed in linear time.

**Example 8.3.1** We shall give an example of how the translation from Fact 8.3.6 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 will 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

$$P(\mathcal{B}_1) \cap P(\mathcal{B}_2) \neq \emptyset \Leftrightarrow (\mathbb{N}, +) \models \exists x_1, \dots, x_k (\varphi_1(\bar{x}) \wedge \varphi_2(\bar{x})).$$

Clearly, we can move the existential quantifiers in  $\varphi_1$  and  $\varphi_2$  to the front of the formula (after introducing new variable names) yielding an existential formula  $\theta$  with  $3k$  quantifiers. That is, the formula  $\theta$  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 8.3.5 immediately follows.

As a final remark, Proposition 8.3.5 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 8.2.1 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 8.3.7** *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 8.3.5, 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 8.2.1 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 8.3.6, 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  $(\mathbb{N}, +) \models \forall \bar{x} \phi(x_1, \dots, x_k)$ , 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.

### 8.3.3 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  $(\mathbb{N}, +) \models \varphi(i_1, \dots, i_k)$ . 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 8.3.8** *Presburger-constrained graph reachability is complete for the class  $STA(*, 2^{2^{n^{O(1)}}}, n)$ .*

To show this proposition, recall that the precise complexity of evaluating Presburger formulas is precisely  $STA(*, 2^{2^{n^{O(1)}}}, n)$ , due to Berman [Ber80] (also see [Koz06]). To derive the upper bound in Proposition 8.3.8, 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  $\varphi(x_{i_1}, \dots, x_{i_r})$ , then the problem reduces to checking whether

$$(\mathbb{N}, +) \models \exists x_1, \dots, x_k (\psi(x_1, \dots, x_k) \wedge \varphi(x_{i_1}, \dots, x_{i_r})),$$

which can be solved by the procedure for evaluating Presburger formulas. This immediately yields the desired upper bound. Hardness for  $STA(*, 2^{2^{n^{O(1)}}}, n)$  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 8.3.8 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 8.3.8 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 8.3.9** *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

$$(\mathbb{N}, +) \models \varphi(i_1, \dots, i_r) \iff (i_1, \dots, i_r) \in P(\mathcal{B}).$$

Therefore, using Fact 8.3.6, 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  $\varphi'$ .

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 8.2.1 and Fact 8.3.6, 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  $\langle \mathfrak{S}, s, t \rangle$  is a positive instance of the problem iff

$$(\mathbb{N}, +) \models \exists x_1, \dots, x_{r+1} (\Psi(x_1, \dots, x_{r+1}) \wedge \varphi'(x_1, \dots, x_r)).$$

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  $\varphi'$  has a fixed size), which can be evaluated in polynomial time by Lenstra-Scarpellini's algorithm [Len83, Sca84] (also see [Grä88]).  $\square$

## 8.4 Applications to model checking

### 8.4.1 LTL with complex fairness

Give direct algorithms to obtain complexity

Use results from earlier section to do LTL model checking with complex fairness constraints. Such fairness can be used to (for example) count the number of times certain labels have been executed.

### 8.4.2 Branching-time logics

EF-logic

**CTL**

We now prove a complementary result that model checking CTL over reversal-bounded counter systems is undecidable. In fact, this already holds for 1-reversal 3-counter systems and very simple CTL formulas.

**Proposition 8.4.1** *Model checking CTL over 1-reversal 3-counter systems is undecidable.*

*Proof.* This result is almost immediate from the undecidability of emptiness problem for deterministic 0-reversal 3-counter systems that *may test equality of the current values of two counters* [ISD<sup>+</sup>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  $((q, \phi(X)), a, (q', i_1, \dots, i_k))$  of the form:

- for each  $j \geq 1$ ,  $i_j \geq 0$ , and
- $\phi(X)$  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
- $\Delta'$  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  $\alpha$  and  $\beta$  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  $\varphi$  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.  $\square$

## 8.5 Other applications

# Chapter 9

## One-counter processes

We discuss results from [GMT09].

### 9.1 Application to weak-bisimilarity checking



# Chapter 10

## Networks of one-counter processes

We discuss results from [To09a].

## **Part III**

### **Epilogue**

# **Chapter 11**

## **Conclusions and Future work**

# Bibliography

- [ABS01] Aurore Annichini, Ahmed Bouajjani, and Mihaela Sighireanu. Trex: A tool for reachability analysis of complex systems. In Gérard Berry, Hubert Comon, and Alain Finkel, editors, *CAV*, volume 2102 of *Lecture Notes in Computer Science*, pages 368–372. Springer, 2001.
- [AHU83] Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. *Data Structures and Algorithms*. Addison-Wesley, 1983.
- [AHV95] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995.
- [AJNS04] Parosh A. Abdulla, Bengt Jonsson, Marcus Nilsson, and Mayank Saxena. A survey of regular model checking. In *In Proc. of CONCUR04*, volume 3170 of *LNCS*, pages 35–48. Springer, 2004.
- [BA06] Mordechai Ben-Ari. *Principles of Concurrent and Distributed Programming*. Addison Wesley, second edition, 2006.
- [Bar07] Vince Barany. *Automatic Presentations of Infinite Structures*. PhD thesis, RWTH Aachen, 2007.
- [BBF<sup>+</sup>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] Alexandre Boudet and Hubert Comon. Diophantine equations, presburger arithmetic and finite automata. In Hélène Kirchner, editor, *CAAP*, volume 1059 of *Lecture Notes in Computer Science*, pages 30–43. Springer, 1996.

- [BCM<sup>+</sup>90] Jerry R. Burch, Edmund M. Clarke, Kenneth L. McMillan, David L. Dill, and L. J. Hwang. Symbolic model checking:  $10^{20}$  states and beyond. In *LICS* [DBL90], pages 428–439.
- [BdRV01] Patrick Blackburn, Maarten de Rijke, and Yde Venema. *Modal Logic*. Cambridge University Press, 2001.
- [BEM97] Ahmed Bouajjani, Javier Esparza, and Oded Maler. Reachability analysis of pushdown automata: Application to model-checking. In Antoni W. Mazurkiewicz and Józef Winkowski, editors, *CONCUR*, volume 1243 of *Lecture Notes in Computer Science*, pages 135–150. Springer, 1997.
- [Ber79] Jean 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] Sébastien Bardin, Alain Finkel, Jérôme Leroux, and Laure Petrucci. Fast: acceleration from theory to practice. *STTT*, 10(5):401–424, 2008.
- [BFLS05] Sébastien Bardin, Alain Finkel, Jérôme Leroux, and Ph. Schnoebelen. Flat acceleration in symbolic model checking. In Peled and Tsay [PT05], pages 474–488.
- [BG04] Achim Blumensath and Erich Grädel. Finite presentations of infinite structures: Automata and interpretations. *Theory Comput. Syst.*, 37(6):641–674, 2004.
- [BG09] Wilmari Bekker and Valentin Goranko. Symbolic model checking of tense logics on rational Kripke models. *LNAI*, 5489:3–21, 2009.
- [BGR10] Vince Barany, Erich Graedel, and Sasha Rubin. Automata-based presentations of infinite structures, 2010.
- [BHLW10] Pablo Barceló, Carlos A. Hurtado, Leonid Libkin, and Peter T. Wood. Expressive languages for path queries over graph-structured data. In Jan Paredaens and Dirk 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] Ahmed Bouajjani, Peter Habermehl, and Tomás Vojnar. Abstract regular model checking. In Rajeev Alur and Doron Peled, editors, *CAV*, volume 3114 of *Lecture Notes in Computer Science*, pages 372–386. Springer, 2004.
- [BJNT00] Ahmed Bouajjani, Bengt Jonsson, Marcus Nilsson, and Tayssir Touili. Regular model checking. In Emerson and Sistla [ES00], pages 403–418.
- [BLN07] Michael Benedikt, Leonid Libkin, and Frank Neven. Logical definability and query languages over ranked and unranked trees. *ACM Trans. Comput. Log.*, 8(2), 2007.
- [Blu99] Achim Blumensath. Automatic structures. Master’s thesis, RWTH Aachen, 1999.
- [Boi99] Bernard Boigelot. *Symbolic Methods for Exploring Infinite State Spaces*. PhD thesis, Université de Liège, 1999.
- [Bou01] Ahmed Bouajjani. Languages, rewriting systems, and verification of infinite-state systems. In Fernando Orejas, Paul G. Spirakis, and Jan van Leeuwen, editors, *ICALP*, volume 2076 of *Lecture Notes in Computer Science*, pages 24–39. Springer, 2001.
- [BW94] Bernard Boigelot and Pierre Wolper. Symbolic verification with periodic sets. In David L. Dill, editor, *CAV*, volume 818 of *Lecture Notes in Computer Science*, pages 55–67. Springer, 1994.
- [Cac02] Thierry Cachat. Uniform solution of parity games on prefix-recognizable graphs. *Electr. Notes Theor. Comput. Sci.*, 68(6), 2002.
- [Cau90] Didier Caucal. On the regular structure of prefix rewriting. In André Arnold, editor, *CAAP*, volume 431 of *Lecture Notes in Computer Science*, pages 87–102. Springer, 1990.
- [Cau92] Didier Caucal. On the regular structure of prefix rewriting. *Theor. Comput. Sci.*, 106(1):61–86, 1992.
- [Cau03] Didier Caucal. On infinite transition graphs having a decidable monadic theory. *Theor. Comput. Sci.*, 290(1):79–115, 2003.

- [CDG<sup>+</sup>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.
- [CGP99] Edmund M. Clarke, Orna Grumberg, and Doron A. Peled. *Model checking*. The MIT Press, 1999.
- [CH90] Kevin J. Compton and C. Ward 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] Marek Chrobak. Finite automata and unary languages. *Theor. Comput. Sci.*, 47(3):149–158, 1986.
- [Cip95] Barry Cipra. How number theory got the best of the pentium chip. *Science*, 267(5195):175, 1995.
- [Col02] Thomas Colcombet. On families of graphs having a decidable first order theory with reachability. In Peter Widmayer, Francisco Triguero Ruiz, Rafael Morales Bueno, Matthew Hennessy, Stephan Eidenbenz, and Ricardo Conejo, editors, *ICALP*, volume 2380 of *Lecture Notes in Computer Science*, pages 98–109. Springer, 2002.
- [DBL90] *Proceedings, Fifth Annual IEEE Symposium on Logic in Computer Science, 4-7 June 1990, Philadelphia, Pennsylvania, USA*. IEEE Computer Society, 1990.
- [DIB<sup>+</sup>00] Zhe Dang, Oscar H. Ibarra, Tevfik Bultan, Richard A. Kemmerer, and Jianwen Su. Binary reachability analysis of discrete pushdown timed automata. In Emerson and Sistla [ES00], pages 69–84.
- [DLS02] Dennis Dams, Yassine Lakhnech, and Martin Steffen. Iterating transducers. *J. Log. Algebr. Program.*, 52-53:109–127, 2002.
- [DT90] Max Dauchet and Sophie Tison. The theory of ground rewrite systems is decidable. In *LICS* [DBL90], pages 242–248.
- [Edm67] Jack Edmonds. Systems of distinct representatives and linear algebra. *J. Res. Nat. Bur. Standards*, 71B:241–245, 1967.

- [EHR00] Javier Esparza, David Hansel, Peter Rossmanith, and Stefan Schwoon. Efficient algorithms for model checking pushdown systems. In Emerson and Sistla [ES00], pages 232–247.
- [EKS03] Javier Esparza, Antonín Kucera, and Stefan Schwoon. Model checking ltl with regular valuations for pushdown systems. *Inf. Comput.*, 186(2):355–376, 2003.
- [ES00] E. Allen Emerson and A. Prasad 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.
- [Esp97] Javier Esparza. Petri nets, commutative context-free grammars, and basic parallel processes. *Fundam. Inf.*, 31(1):13–25, 1997.
- [EVW02] Kousha Etessami, Moshe Y. Vardi, and Thomas Wilke. First-order logic with two variables and unary temporal logic. *Inf. Comput.*, 179(2):279–295, 2002.
- [FL02] Alain Finkel and Jérôme Leroux. How to compose presburger-accelerations: Applications to broadcast protocols. In Manindra Agrawal and Anil Seth, editors, *FSTTCS*, volume 2556 of *Lecture Notes in Computer Science*, pages 145–156. Springer, 2002.
- [GJ79] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness*. W.H.Freeman & Co Ltd, 1979.
- [GMT09] Stefan Göller, Richard Mayr, and Anthony Widjaja To. On the computational complexity of verifying one-counter processes. In *LICS*, pages 235–244. IEEE Computer Society, 2009.
- [GN08] Wouter Gelade and Frank Neven. Succinctness of the complement and intersection of regular expressions. In Susanne Albers and Pascal Weil, editors, *STACS*, volume 1 of *LIPICs*, pages 325–336. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany, 2008.
- [Göl08] Stefan Göller. Reachability on prefix-recognizable graphs. *Inf. Process. Lett.*, 108(2):71–74, 2008.



- [Grä88] Erich Grädel. Subclasses of presburger arithmetic and the polynomial-time hierarchy. *Theor. Comput. Sci.*, 56:289–301, 1988.
- [Grä90] Erich Grädel. Simple interpretations among complicated theories. *Inf. Process. Lett.*, 35(5):235–238, 1990.
- [Gre09] Samuel Greengard. Making automation work. *Commun. ACM*, 52(12):18–19, 2009.
- [GS66] Seymour Ginsburg and Edwin 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.
- [GS00] Susanne Graf and Michael 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.
- [Har86] David Harel. Effective transformations on infinite trees, with applications to high undecidability, dominoes, and fairness. *J. ACM*, 33(1):224–248, 1986.
- [Hod83] Bernard R. Hodgson. Decidabilite par automate fini. *Ann. sc. math. Quebec*, 7(1):39–57, 1983.
- [hom] LASH homepage. <http://www.montefiore.ulg.ac.be/~boigelot/research/lash/>.
- [HPW09a] Carlos A. Hurtado, Alexandra Poulouvasilis, and Peter T. Wood. Finding top- approximate answers to path queries. In Troels Andreasen, Ronald R. Yager, Henrik Bulskov, Henning Christiansen, and Henrik Legind Larsen, editors, *FQAS*, volume 5822 of *Lecture Notes in Computer Science*, pages 465–476. Springer, 2009.
- [HPW09b] Carlos A. Hurtado, Alexandra Poulouvasilis, and Peter T. Wood. Ranking approximate answers to semantic web queries. In Lora Aroyo, Paolo

Traverso, Fabio Ciravegna, Philipp Cimiano, Tom Heath, Eero Hyvönen, Riichiro Mizoguchi, Eyal Oren, Marta Sabou, and Elena Paslaru Bon-tas Simperl, editors, *ESWC*, volume 5554 of *Lecture Notes in Computer Science*, pages 263–277. Springer, 2009.

- [Huy83] Dung T. Huynh. Commutative grammars: The complexity of uniform word problems. *Information and Control*, 57(1):21–39, 1983.
- [Iba78] Oscar H. Ibarra. Reversal-bounded multicounter machines and their decision problems. *J. ACM*, 25(1):116–133, 1978.
- [IK89] Neil Immerman and Dexter Kozen. Definability with bounded number of bound variables. *Inf. Comput.*, 83(2):121–139, 1989.
- [ISD<sup>+</sup>02] Oscar H. Ibarra, Jianwen Su, Zhe Dang, Tevfik Bultan, and Richard A. Kemmerer. Counter machines and verification problems. *Theor. Comput. Sci.*, 289(1):165–189, 2002.
- [JM00] Daniel Jurafsky and James H. Martin. *Speech and Language Processing*. Prentice Hall, 2000.
- [JN00] Bengt Jonsson and Marcus Nilsson. Transitive closures of regular relations for verifying infinite-state systems. In Graf and Schwartzbach [GS00], pages 220–234.
- [Joh86] J. Howard Johnson. Rational equivalence relations. *Theor. Comput. Sci.*, 47(3):39–60, 1986.
- [Kam68] Hans Kamp. *Tense Logic and the Theory of Linear Order*. PhD thesis, UCLA, 1968.
- [Kan83] Ravi Kannan. Improved algorithms for integer programming and related lattice problems. In *STOC*, pages 193–206. ACM, 1983.
- [Kla08] Felix Klaedtke. Bounds on the automata size for presburger arithmetic. *ACM Trans. Comput. Log.*, 9(2), 2008.
- [KMM<sup>+</sup>97] Yonit Kesten, Oded Maler, Monica Marcus, Amir Pnueli, and Elad Shahar. Symbolic model checking with rich assertion languages. In Orna Grumberg, editor, *CAV*, volume 1254 of *Lecture Notes in Computer Science*, pages 424–435. Springer, 1997.

- [KMM<sup>+</sup>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] Eryk Kopczynski. Complexity of problems for commutative grammars. *CoRR*, abs/1003.4105, 2010.
- [Koz97] Dexter C. Kozen. *Automata and Computability*. Springer-Verlag, 1997.
- [Koz06] Dexter C. Kozen. *Theory of Computation*. Springer-Verlag, 2006.
- [KPV02] Orna Kupferman, Nir Piterman, and Moshe Y. Vardi. Model checking linear properties of prefix-recognizable systems. In Ed Brinksma and Kim Guldstrand Larsen, editors, *CAV*, volume 2404 of *Lecture Notes in Computer Science*, pages 371–385. Springer, 2002.
- [KRS05] Bakhadyr Khoussainov, Sasha Rubin, and Frank Stephan. Automatic linear orders and trees. *ACM Trans. Comput. Log.*, 6(4):675–700, 2005.
- [KT10] Eryk Kopczynski and Anthony Widjaja To. Parikh images of grammars: Complexity and applications. *To appear in LICS*, 2010.
- [Len83] H.W. Lenstra. Integer programming with a fixed number of variables. *Math. Oper. Res.*, 8(4):538–548, 1983.
- [Lib04] Leonid Libkin. *Elements Of Finite Model Theory (Texts in Theoretical Computer Science. An EATCS Series)*. Springer-Verlag, 2004.
- [Lö03] Christof Löding. *Infinite Graphs Generated by Tree Rewriting*. PhD thesis, RWTH Aachen, 2003.
- [Lö06] Christof Löding. Reachability problems on regular ground tree rewriting graphs. *Theory Comput. Syst.*, 39(2):347–383, 2006.
- [LS02] Denis Lugiez and Ph. Schnoebelen. The regular viewpoint on pa-processes. *Theor. Comput. Sci.*, 274(1-2):89–115, 2002.
- [LS04] Jérôme Leroux and Grégoire Sutre. On flatness for 2-dimensional vector addition systems with states. In Philippa Gardner and Nobuko Yoshida, editors, *CONCUR*, volume 3170 of *Lecture Notes in Computer Science*, pages 402–416. Springer, 2004.

- [LS05a] Jérôme Leroux and Grégoire Sutre. Flat counter automata almost everywhere! In Peled and Tsay [PT05], pages 489–503.
- [LS05b] Denis Lugiez and Ph. Schnoebelen. Decidable first-order transition logics for pa-processes. *Inf. Comput.*, 203(1):75–113, 2005.
- [Lyn96] Nancy A. Lynch. *Distributed Algorithms*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1996.
- [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.
- [May98] Richard Mayr. *Decidability and Complexity of Model Checking Problems for Infinite-State Systems*. PhD thesis, TU-Munich, 1998.
- [McM93] Kenneth L. McMillan. *Symbolic Model Checking: An Approach to the State Explosion Problem*. Kluwer Academic Publishers, 1993.
- [Min67] M. L. Minsky. *Computation: Finite and Infinite Machines*. Prentice-Hall, Englewood Cliff, NJ, 1967.
- [Mor00] Christophe Morvan. On rational graphs. In *FOSSACS '00*, pages 252–266, 2000.
- [MP71] M. McNaughton and S. Papert. *Counter-Free Automata*. MIT Press, 1971.
- [MS85] David E. Muller and Paul E. Schupp. The theory of ends, pushdown automata, and second-order logic. *Theor. Comput. Sci.*, 37:51–75, 1985.
- [MW95] Alberto O. Mendelzon and Peter T. Wood. Finding regular simple paths in graph databases. *SIAM J. Comput.*, 24(6):1235–1258, 1995.
- [Nil05] Marcus Nilsson. *Regular Model Checking*. PhD thesis, Uppsala Universitet, 2005.
- [Pap81] Christos H. Papadimitriou. On the complexity of integer programming. *J. ACM*, 28(4):765–768, 1981.
- [Par66] Rohit J. Parikh. On context-free languages. *J. ACM*, 13(4):570–581, 1966.

- [PS98] C.H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Dover Publications, 1998.
- [PSwW02] Giovanni Pighizzini, Jeffrey Shallit, and Ming wei 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] Doron Peled and Yih-Kuen 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.
- [Rab70] Michael 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.
- [Rub08] Sasha Rubin. Automata presenting structures: A survey of the finite string case. *Bulletin of Symbolic Logic*, 14(2):169–209, 2008.
- [Sca84] Bruno 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 Philippe Balbiani, Nobu-Yuki Suzuki, Frank Wolter, and Michael Zakaryashev, editors, *Advances in Modal Logic*, pages 393–436. King’s College Publications, 2002.
- [Sip97] Michael Sipser. *Introduction to the Theory of Computation*. PWS Publishing Company, 1997.
- [SM73] Larry J. Stockmeyer and Albert 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] Helmut Seidl, Thomas Schwentick, Anca Muscholl, and Peter Habermehl. Counting in trees for free. In Josep Díaz, Juhani Karhumäki, Arto Lepistö, and Donald Sannella, editors, *ICALP*, volume 3142 of *Lecture Notes in Computer Science*, pages 1136–1149. Springer, 2004.

- [Sti01] Colin Stirling. *Modal and temporal properties of processes*. Springer-Verlag New York, Inc., New York, NY, USA, 2001.
- [Sto74] Larry J. Stockmeyer. *The complexity of decision problems in automata theory and logic*. PhD thesis, Department of Electrical Engineering, MIT, 1974.
- [Tho96] Wolfgang Thomas. Languages, automata, and logic. In *Handbook of Formal Languages*, pages 389–455. Springer, 1996.
- [Tho03] Wolfgang Thomas. Constructing infinite graphs with a decidable mso-theory. In Branislav Rován and Peter Vojtáš, editors, *MFCS*, volume 2747 of *Lecture Notes in Computer Science*, pages 113–124. Springer, 2003.
- [TL08] Anthony Widjaja To and Leonid 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] Anthony Widjaja To and Leonid Libkin. Algorithmic metatheorems for decidable LTL model checking over infinite systems. In C.-H. Luke Ong, editor, *FOSSACS*, volume 6014 of *Lecture Notes in Computer Science*, pages 221–236. Springer, 2010.
- [To09a] Anthony Widjaja To. Model checking FO(R) over one-counter processes and beyond. In Erich Grädel and Reinhard Kahle, editors, *CSL*, volume 5771 of *Lecture Notes in Computer Science*, pages 485–499. Springer, 2009.
- [To09b] Anthony Widjaja To. Unary finite automata vs. arithmetic progressions. *Inf. Process. Lett.*, 109(17):1010–1014, 2009.
- [To10] Anthony Widjaja To. Parikh images of regular languages: Complexity and applications. *CoRR*, abs/1002.1464, 2010.
- [Var95] Moshe Y. Vardi. An automata-theoretic approach to linear temporal logic. In Faron Moller and Graham M. Birtwistle, editors, *Banff Higher Order Workshop*, volume 1043 of *Lecture Notes in Computer Science*, pages 238–266. Springer, 1995.

- [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] Kumar Neeraj Verma, Helmut Seidl, and Thomas Schwentick. On the complexity of equational horn clauses. In Robert Nieuwenhuis, editor, *CADE*, volume 3632 of *Lecture Notes in Computer Science*, pages 337–352. Springer, 2005.
- [VW86a] Moshe Y. Vardi and Pierre Wolper. An automata-theoretic approach to automatic program verification (preliminary report). In *LICS*, pages 332–344. IEEE Computer Society, 1986.
- [VW86b] Moshe Y. Vardi and Pierre Wolper. Automata-theoretic techniques for modal logics of programs. *Journal of Computer and System Sciences*, 32:183–221, 1986.
- [Wal00] Igor Walukiewicz. Model checking CTL properties of pushdown systems. In Sanjiv Kapoor and Sanjiva Prasad, editors, *FSTTCS*, volume 1974 of *Lecture Notes in Computer Science*, pages 127–138. Springer, 2000.
- [WB98] Pierre Wolper and Bernard Boigelot. Verifying systems with infinite but regular state spaces. In Alan J. Hu and Moshe Y. Vardi, editors, *CAV*, volume 1427 of *Lecture Notes in Computer Science*, pages 88–97. Springer, 1998.
- [WB00] Pierre Wolper and Bernard Boigelot. On the construction of automata from linear arithmetic constraints. In Graf and Schwartzbach [GS00], pages 1–19.
- [Wol00] Pierre Wolper. Constructing automata from temporal logic formulas: A tutorial. In Ed Brinksma, Holger Hermanns, and Joost-Pieter 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] Stefan Wöhrle and Wolfgang Thomas. Model checking synchronized products of infinite transition systems. *Logical Methods in Computer Science*, 3(4), 2007.

- [YKB09] Tuba Yavuz-Kahveci and Tevfik Bultan. Action language verifier: an infinite-state model checker for reactive software specifications. *Formal Methods in System Design*, 35(3):325–367, 2009.
- [YKBB05] Tuba Yavuz-Kahveci, Constantinos Bartzis, and Tevfik Bultan. Action language verifier, extended. In Kousha Etessami and Sriram 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  $\alpha_i$  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  $\varepsilon$ . 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  $\alpha_{k+1}$ ,  $\beta_k$ ,  $\eta_{k+1}$ , and  $\theta_{k+1}$ . 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  $\eta_{k+1}$  to be the prefix of length  $|v'|$  of the run  $\rho$  of  $\mathcal{A}^{q_k}$  on  $v'v''$  given by Proposition 4.1.5. We define  $\theta_{k+1}$  to be the run  $\rho'$  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  $q_{k+1} := \text{last}(\rho_{k+1})$  and  $q'_{k+1} := \text{last}(\rho'_{k+1})$ . 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

$$D = (D_1 \cap D_2) \cup \{vi \in D_1 \setminus D_2 : i \in Y, v \in D_2\}.$$

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  $\tau$  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  $i \in Y$ . 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  $i \in Y$  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,  $\tau$  and  $\tau'$  coincide except when evaluated on the context leaves. This completes the proof of uniqueness.

# Appendix B

## Proofs from Chapter 8

### B.1 Proof of Fact 8.2.3

The proof is by induction on the length  $|\pi|$  of the path  $\pi$ . 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 8.2.3 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  $\pi$  be a path of length  $k$ . If  $\pi$  is simple, then we may set  $\pi' := \pi$  and  $h := 0$ , and the proof is complete. Therefore, assume that  $\pi$  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  $\pi$  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  $\pi$ . We will now apply the induction hypothesis twice. Firstly, applying the induction hypothesis on the path  $\pi_1$ , we obtain a simple path  $\pi'$  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 8.2.3 to value  $k$ . Therefore, by mathematical induction, Fact 8.2.3 holds for all values of  $k$ , which completes the proof.

## B.2 Proof of Lemma 8.2.5

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  $\pi$  as follows. Let  $t$  be the first position where the letter  $a_i$  occurs in  $\pi$ , 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.

## B.3 Proof of Lemma 8.2.6

This can be done using Lemma 8.2.5 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  $2^{O(k \log n)}$ .

## B.4 Proof of Proposition 8.2.10

The automaton  $\mathcal{A}_n = (\Sigma_n, Q_n, \delta_n, q_0, q_F)$ , where

$$\begin{aligned} Q_n &= \{q_0, q_F\} \cup \{p_0, \dots, p_n\} \cup \{p'_1, \dots, p'_n\}, \\ \Sigma_n &= \{a, b, c\} \cup \{a_i, a'_i : 1 \leq i \leq n\}. \end{aligned}$$

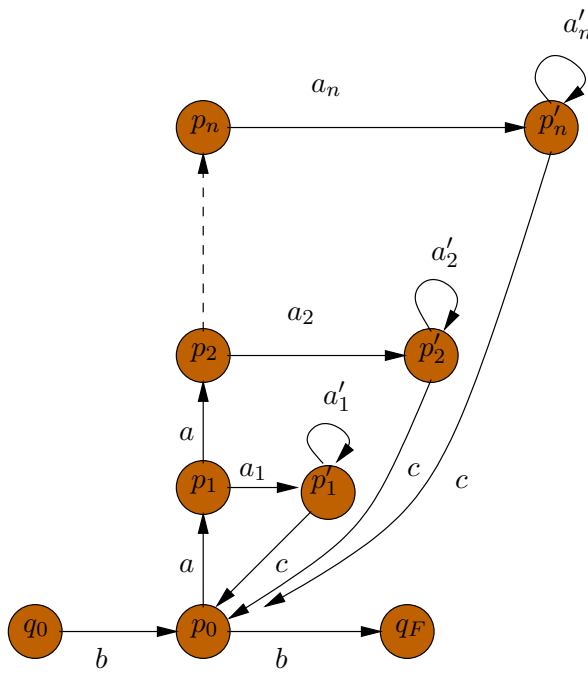


Figure B.1: A depiction of the DWA  $\mathcal{A}_n$

We specify the transition function  $\delta_n$  in Figure B.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  $\pi$ .

**Claim B.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  $\alpha$ -component of  $\mathbf{u}_{j_i}$ , where  $\alpha \neq a'_i$  for

all  $i \in \{1, \dots, n\}$ , is 0 as each such letter  $a_i$  occurs at most  $n(n+1)/2$  times in  $\pi$ . 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.

## B.5 Proof of Proposition 8.3.2

### 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  $\phi$  in 3-CNF, does there exist a satisfying assignment for  $\phi$  that additionally makes no more than one literal true for each clause. We shall call such an 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  $\varphi$  is a positive instance of one-in-three 3SAT iff  $\mathbf{1} \in \mathcal{P}(\mathcal{L}(e_\varphi))$ . To prove this, suppose that  $\varphi$  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  $\sigma$  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  $e_\varphi$  and  $\mathbf{1}$  can be done in time polynomial in the size of  $\varphi$ .