Università degli Studi di Milano
Doctor of Philosophy in Computer Science
Department of Computer Science

Dottorato di Ricerca in Informatica - Ciclo XXVII

Descriptional Complexity and Parikh Equivalence

Settore Scientifico Disciplinare INF/01 Informatica

Tesi di Dottorato di Ricerca di:
Giovanna Janet Lavado

Supervisor:
Prof. Giovanni Pighizzini

Co-supervisor:
Prof. Carlo Mereghetti

Coordinator:
Prof. Ernesto Damiani

A.A. 2013 - 2014

To my husband, Angelo.

Acknowledgements

I want to thank my supervisors, Giovanni Pighizzini and Carlo Mereghetti, for the opportunity to work with them in this thesis.

I would also like to thank the reviewers, Flavio D'alessandro, Viliam Geffert, Martin Kutrib and Shinnosuke Seki for their precious work of revision, full of useful comments, suggestions and, of course, corrections.

Finally, I want to thank my friends and colleagues for their friendship and support during these years of Ph.D.

Quiero agradecer también a mis padres por todo el apoyo incondicional que me han brindado.

Abstract

This thesis was carried out in the G.-C. Rota Laboratory of Languages and Combinatorics (LIN.COM), at the Dipartimento di Informatica, Università degli Studi di Milano. The thesis deals with some topics in the theory of formal languages and automata. Specifically, the thesis deals with the theory of context-free languages and the study of their descriptional complexity.

The descriptional complexity of a formal structure (e.g., grammar, model of automata, etc) is the number of symbols needed to write down its description. While this aspect is extensively treated in regular languages, as evidenced by numerous references, in the case of context-free languages few results are known.

An important result in this area is the Parikh's theorem. The theorem states that for each context-free language there exists a regular language with the same Parikh image. Given an alphabet $\Sigma = \{a_1, \ldots, a_m\}$, the Parikh image is a function $\psi : \Sigma^* \to \mathbb{N}^m$ that associates with each word $w \in \Sigma^*$, the vector $\psi(w) = (|w|_{a_1}, |w|_{a_2}, \ldots, |w|_{a_m})$, where $|w|_{a_i}$ is the number of occurrences of a_i in w. The Parikh image of a language $L \subseteq \Sigma^*$ is the set of Parikh images of its words. For instance, the language $\{a^n b^n \mid n \geq 0\}$ has the same Parikh image as $(ab)^*$. Roughly speaking, the theorem shows that if the order of the letters in a word is disregarded, retaining only the number of their occurrences, then context-free languages are indistinguishable from regular languages.

Due to the interesting theoretical property of the Parikh's theorem, the goal of this thesis is to study some aspects of descriptional complexity according to Parikh equivalence. In particular, we investigate the conversion of one-way nondeterministic finite automata and context-free grammars into Parikh equivalent one-way and two-way deterministic finite automata, from a descriptional complexity point of view.

We prove that for each one-way nondeterministic automaton with n states there exist Parikh equivalent one-way and two-way deterministic automata with $e^{O(\sqrt{n \cdot \ln n})}$ and $p(n)$ states, respectively, where $p(n)$ is a polynomial. Furthermore, these costs are tight. In contrast, if all the words accepted by the given one-way nondeterministic automaton contain at least two different letters, then a Parikh equivalent one-way deterministic automaton with a polynomial number of states can be found.

Concerning context-free grammars, we prove that for each grammar in Chomsky normal form with h variables there exist Parikh equivalent one-way and two-way deterministic automata with $2^{O(h^2)}$ and $2^{O(h)}$ states, respectively. Even these bounds are tight.

A further investigation is the study under Parikh equivalence of the state complexity of some language operations which preserve regularity. For union, concatenation, Kleene star, complement, intersection, shuffle, and reversal, we obtain a polynomial state complexity over any fixed alphabet, in contrast to the intrinsic exponential state complexity of some of these operations in the classical version. For projection we prove a superpolynomial state complexity, which is lower than the exponential one of the corresponding classical operation. We also prove that for each two one-way deterministic automata A and B it is possible to obtain a one-way deterministic automaton with a polynomial number of states whose accepted language has as Parikh image the intersection of the Parikh images of the languages accepted by A and B.

Keywords: automata and formal languages, context-free languages, semilinear set, Parikh image, descriptional complexity.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

L'Apprenti sorcier

Paul Dukas

This thesis deals with the theory of descriptional complexity and Parikh equivalence. Descriptional complexity is an area of theoretical computer science in which one of the main questions is how succinctly a formal language can be described by a formalism in comparison with other formalisms. For instance, a fundamental result in this area is the exponential trade-off between nondeterministic and deterministic finite automata with respect to the number of states.

It is well known that the state cost of the conversion of one-way nondeterministic finite automata (1nfas) into equivalent one-way deterministic finite automata (1dfas) is exponential: using the classical subset construction [RS59], from each n-state 1nfa we can build an equivalent 1dfa with 2^n states. Furthermore, this cost cannot be reduced in the worst case.

In all examples witnessing such a state gap (e.g., [Lup63, MF71, Moo71]), input alphabets with at least two letters and proof arguments strongly relying on the structure of words are used. As a matter of fact, for the unary case, namely the case of the one letter input alphabet, the cost reduces to $e^{\Theta(\sqrt{n \cdot \ln n})}$, as shown by Chrobak [Chr86]. So, we can ask:

> What happens if we do not care of the order of symbols in the words, i.e., if we are interested only in obtaining 1dfas accepting sets of words which are equal, after permuting the symbols, to the words accepted by the given 1nfas?

This question is related to the well-known notions of Parikh image and Parikh equivalence [Par66], which have been extensively investigated in the literature (e.g., [Gol77, AÉI02]) even for the connections of semilinear sets [Huy80] and with other fields of investigation as, e.g., Presburger arithmetics [GS66], Petri nets [Esp97], logical formulas [VSS05] and formal verification [To10a].

The goal of the thesis is to study some aspects of descriptional complexity according to Parikh equivalence.

We remind the reader that two words over a same alphabet Σ are Parikh equivalent if they are equal up to a permutation of their symbols or, equivalently, for each letter $a \in \Sigma$, the number of occurrences of a in the two words is the same. This notion extends in a natural way to languages and to formal systems which are used to specify languages as, for instance, grammars and automata. Two languages are Parikh equivalent if they are the same language unless the order of the symbols in the words is disregarded.

An important result in this topic is the Parikh theorem, which states that every context-free language is Parikh equivalent to a regular language. For instance, the language $\{a^n b^n \mid n \geq 0\}$ is Parikh equivalent to regular language $(ab)^*$. Intuitively, the theorem shows that if the order of the letters in a word is disregarded, retaining only the number of their occurrences, then context-free languages are indistinguishable from regular languages.

Notice that in the unary case Parikh equivalence is just the standard equivalence. So, in the unary case, the answer to our previous question is given by the above mentioned result by Chrobak [Chr86].

Our first contribution in this thesis is an answer to that question in the general case. In particular, we prove that the state cost of the conversion of n-state 1nfas into Parikh equivalent 1dfas is the same as in the unary case, i.e., it is $e^{\Theta(\sqrt{n \cdot \ln n})}$. More surprisingly, we prove that the worst case is due to the unary words accepted by the automaton. In fact, we show that if the given 1nfa accepts only nonunary words, i.e., each accepted word contains at least two different letters, then we can obtain a Parikh equivalent 1dfa with a polynomial number of states in n. Hence, while in the standard determinization, the most difficult part of the conversion, with respect to the state complexity, comes from the nonunary part of a language, in the "Parikh determinization" this part becomes easy and the most complex part is the unary one.

In the second part of the thesis we consider context-free grammars (cfgs). Parikh Theorem [Par66] states that each context-free language is Parikh equivalent to a regular language. We study this equivalence from a descriptional complexity point of view. Recently, Esparza, Ganty, Kiefer, and Luttenberger proved that each cfg G in Chomsky normal form with h variables can be converted into a Parikh equivalent 1nfa with $O(4^h)$ states [EGKL11]. In [LP12] it was proved that if G generates a bounded language then we can obtain a 1dfa with $2^{h^{O(1)}}$ states, i.e., a number exponential in a polynomial of the number of variables. In this thesis, we are able to extend such a result by removing the restriction to bounded languages. We also reduce the upper bound to $2^{O(h^2)}$. A milestone for obtaining this result is the conversion of 1nfas to Parikh equivalent 1dfas presented in the first part of the thesis. By suitably combining that conversion (in particular the polynomial conversion in the case of 1nfas accepting nonunary words) with the above mentioned result from [EGKL11] and with a result by Pighizzini, Shallit, and Wang [PSW02] concerning the unary case, we prove that each context-free grammar in Chomsky normal form with h variables can be converted into a Parikh equivalent 1dfa with $2^{O(h^2)}$ states. From the results concerning the unary case, it follows that this bound is tight. Even for this simulation, as for that of 1nfas by Parikh equivalent 1dfas, the main contribution to the state complexity of the resulting automaton is given by the unary part.

In the third part of the thesis, we consider conversions of 1nfas and cfgs into Parikh equivalent *two-way deterministic automata* (2dfas). Due to the fact that in the unary case these conversions are less expensive than the corresponding ones into 1dfas, we are able to prove that each n-state 1nfa can be converted into an equivalent 2dfa with a number of states polynomial in n, and each context-free grammar in Chomsky normal form with h variables can be converted into a Parikh equivalent 2dfa with a number of states exponential in h.

The investigation of the state complexity of regular languages and their operations is extensively reported in the literature (e.g., [PS02, Yu00, YZS94]). Motivated by the interest in regular languages, we continue the same line of research by considering basic operations on regular languages and on 1dfas. In the fourth part of the thesis we reformulate under Parikh equivalence some classical questions on the state complexity of operations as, for instance, the following: given two arbitrary 1dfas A and B of n_1 and n_2 states, respectively, how many states are sufficient and necessary in the worst case (as a function of n_1 and n_2) for a 1dfa to accept the concatenation of the languages accepted by A and B? For this question an exponential cost is known [YZS94]. Using our above mentioned bound on the conversion of 1nfas into Parikh equivalent 1dfas, this exponential bound can be reduced, under Parikh equivalence, to a superpolynomial upper bound. In this thesis we further reduce it to a polynomial, namely we show that there exists a 1dfa with a number of states polynomial in n_1 and n_2 accepting a language that is Parikh equivalent to the concatenation of the languages accepted by A and B. We obtain a similar result for the Kleene star operation while, for the union, the cost is polynomial even in the classical case. We also present results for other operations as intersection, complement, reversal, shuffle and projection.

Concerning intersection and complement, we observe that these operations do not commute with Parikh image, e.g., the Parikh image of the complement of a language L does not necessarily coincide with the complement of the Parikh image of L. However, semilinear sets are closed under intersection and complement [GS64]. Hence, we can formulate state complexity questions about intersection and complement of Parikh images of languages accepted by given 1dfas. We solve the question for the intersection by proving, in a constructive way, that for each two 1dfas there exists a 1dfa of polynomial size accepting a language whose Parikh image is the intersection of the Parikh images of the languages accepted by the two given 1dfas, while the analogous question for complement will be the subject of future investigations.

The study of conversions into Parikh equivalent one-way deterministic automata has been published in [LPS12], while the study of conversions into Parikh equivalent two-way deterministic automata has been published in [LPS13]. The investigation of operational state complexity of one-way deterministic automata under Parikh equivalence has been published in [LPS14].

1.1 Motivations

The main motivation of my thesis is the study of how succinctly a formal language can be described by a formalism in comparison with other formalisms. More precisely, we study the succinctness of these formalisms such as finite automata, pushdown automata and context-free grammars. In other words, the goal of this research is to find "small devices". This brings us to the area of descriptional complexity (often called conciseness, succinctness, or economy of description) [GKK$^+$02], that is, the science or sometimes the art of making the description of objects as simple as possible and only as complex as necessary.

My contribution to this field is through the use of the Parikh theorem. Just to remind how it works, the Parikh mapping, introduced by Parikh [Par66], provides a connection between formal language theory and number theory. It associates a word with the vector of natural numbers that reflects the number of the occurrences of the symbols in this word. Intuitively, the Parikh theorem states that the class of context-free languages, with respect to the Parikh images does not differ from the class of regular languages. Moreover, Parikh succeeded in characterizing the Parikh images of context-free and thus also of regular languages, namely in the framework of semilinear sets.

In addition of what will be exposed in this thesis, semilinear sets have been extensively studied in many other important areas. For example, Ginsburg and Spanier [GS64, GS66] have shown that the class of semilinear sets enjoys many good properties, not only with respect to closure properties, but also with respect to decision properties. Many good properties of semilinear sets justify their application in solving many decision problems in formal language theory. Most prominently, the decidability of the emptiness problem for a language immediately follows once we have shown that the Parikh image of this language effectively yields a semilinear set. For example, the decidability of the emptiness problem for context-free languages immediately follows from Parikh theorem.

Semilinear sets, by themselves of purely number-theoretical nature, are therefore tightly connected with formal language theory. In addition, the notion of semilinear sets is related to logic. The class of semilinear sets turned out to coincide with the class of sets definable in Presburger arithmetic. Presburger has introduced in [Pre29, PJ91] the first-order theory of the integer numbers with addition and ordering relation $(\mathbb{Z}, 0, 1, +, <)$, called Presburger arithmetic. Presburger showed that this theory is complete and therefore decidable. This connection, in fact, confirms the many good properties of semilinear sets.

Presburger arithmetic and semilinear sets play an important role in current research. In the field of system verification, on the one hand, many decision problems concerning model-checking infinite-state systems can be reduced to the emptiness problem for semilinear sets. Examples of such infinite-state systems are discrete-timed pushdown automata (see Dang et al. [Dan01]), several extensions of reversal-bounded counter machines (see Ibarra et al. [IBS00]), and some classes of semilinear systems (see Bouajjani and Habermehl [BH96]).

Other use of Parikh theorem can be found in the Petri nets area. For example, a structural characterization of the reachable markings of Petri nets has been shown in [Esp97]. They obtain a structural characterization of the set of reachable markings of communication-free nets, and use it to prove that the reachability problem for this class is NP-complete. Another consequence of the characterization is that the set of reachable markings of communication-free nets is effectively a semilinear set.

1.2 Contributions and outline

Summarizing, our contributions are the followings:

1. Conversions of one-way nondeterministic finite automata and context-free grammars into Parikh equivalent one-way deterministic finite automata.

2. In particular, one of our main contributions is the conversion from one-way nondeterministic finite automata accepting only nonunary words into Parikh equivalent one-way deterministic finite automata. This result is important for our constructions.

3. Conversions of one-way nondeterministic finite automata and context-free grammars into Parikh equivalent two-way deterministic finite automata.

4. We investigate under Parikh equivalence the state complexity of some language operations which preserve regularity.

5. The noncommutativity of intersection of two languages with respect to Parikh image.

Finally, let me introduce the outline of my thesis.

In Chapter 2 we fix our notation. Moreover, we present basic facts concerning formal languages and automata.

In Chapter 3 we recall the notion of descriptional complexity. Furthermore, we present preliminary constructions of composition and decomposition of regular and context-free languages.

In Chapter 4 we present basic facts concerning semilinear sets and Parikh theorem.

In Chapter 5 we study conversions under Parikh equivalence. More precisely, in Section 5.1 we study conversions of one-way nondeterministic finite automata and context-free grammars into Parikh equivalent one-way deterministic finite automata, while in Section 5.2 we extend the results to conversions into Parikh equivalent two-way deterministic finite automata.

In Chapter 6 we study the operational state complexity of one-way deterministic automata under Parikh equivalence.

In Chapter 7 we present the conclusions and further works.

Chapter 2

Languages, grammars and automata

Ouverture

Guillaume Tell
Gioacchino Rossini

In this chapter, we introduce some notation that will be used in the sequel, and review basic definitions and results from formal languages and automata theory. For a detailed exposition, we refer the reader to [HU79, Sha08].

2.1 General notation

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

2.1.1 Some notation from set theory

A set is a collection of elements chosen from some domain. If S is a finite set, we use the notation $|S|$ to denote the number of elements or cardinality of the set and 2^S to denote the family of all its subsets. The empty set is denoted by $\emptyset$. By $A \cup B$ (respectively $A \cap B$, $A \setminus B$) we mean the union of the two sets A and B (respectively intersection and set difference). The notation A^c means the complement of the set A with respect to some assumed universal set U, that is, $A^c = \{x \in U \mid x \notin A\}$.

2.1.2 Sets of numbers and vector spaces

We denote the set of integers by $\mathbb{Z} = \{\ldots, -2, -1, 0, 1, 2, \ldots\}$ and the set of nonnegative integers by $\mathbb{N} = \{0, 1, 2, 3, \ldots\}$. Then $\mathbb{Z}^m$ and $\mathbb{N}^m$ denote the corresponding sets of m-dimensional integer vectors including the *null vector* $\mathbf{0} = (0, 0, \ldots, 0)$. For $1 \le i \le m$, we denote the i-th component of a vector $\boldsymbol{v}$ by $\boldsymbol{v}[i]$. Let $\alpha = \{i_1, \ldots, i_k\} \subseteq \{1, \ldots, m\}$. We denote by $\pi_\alpha : \mathbb{N}^m \to \mathbb{N}^k$ the α-projection defined by $\pi_\alpha(x_1, \ldots, x_m) = (x_{i_1}, \ldots, x_{i_k})$.

2.1.3 Asymptotic notation

We use the following standard asymptotic notation, especially when measuring the state complexity: big-O $O()$, big-omega $\Omega()$, big-theta $\Theta()$ and little-o $o()$.

Let f and g be two functions defined on some subset of the real numbers $\mathbb{R}$. We write $f(n) = O(g(n))$ if $\exists c > 0, n_0 \in \mathbb{N}$ such that $\forall n > n_0$,

$$f(n) \le c \cdot g(n)$$

Intuitively, it means that f is asymptotically bounded from above by g (up to constant factor).

We write $f(n) = \Omega(g(n))$ if $\exists c > 0, n_0 \in \mathbb{N}$ such that $\forall n > n_0$,

$$f(n) \ge c \cdot g(n)$$

Intuitively, it means that f is asymptotically bounded from below by g.

We write $f(n) = \Theta(g(n))$ if $\exists c, d > 0, n_0 \in \mathbb{N}$ such that $\forall n > n_0$,

$$c \cdot g(n) \le f(n) \le d \cdot g(n)$$

Intuitively, it means that f is bounded both above and below by g asymptotically. In fact, $f(n) = \Theta(g(n))$ if and only if $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$.

We write $f(n) = o(g(n))$ if

$$lim_{n \to \infty} f(n)/g(n) = 0$$

Intuitively, it means that $g(n)$ grows asymptotically faster than $f(n)$. We can observe that every function that is little-o of g is also big-O of g, but not vice versa.

2.2 Alphabets, strings and languages

An *alphabet* is a finite, nonempty set of symbols. Conventionally, we use the symbol Σ for an alphabet. An alphabet is said to be *unary* when it consists of a single symbol.

A *string* or *word* over Σ is a finite sequence of symbols from Σ. The *empty string* ε is the string with zero symbols. A word is said to be *unary* if it consists only of $k \ge 0$ occurrences of a same symbol, otherwise it is said to be *nonunary*.

The set of all strings over an alphabet Σ is conventionally denoted by Σ^*. The set of nonempty strings from an alphabet Σ is denoted by Σ^+.

Given a word $w \in \Sigma^*$, $|w|$ denotes its *length* namely the number of symbols in it and, for a letter $a \in \Sigma$, $|w|_a$ denotes the *number of occurrences* of a in w.

Let x and y be strings. Then xy denotes the *concatenation* of x and y, that is, if x is composed of n symbols $x = a_1a_2 \cdots a_n$ and y is composed of m symbols $y = b_1b_2 \cdots b_m$, then xy is the string $a_1a_2 \cdots a_nb_1b_2 \cdots b_m$ of length $n + m$.

A *factor* of a word $x \in \Sigma^*$ is a word $y \in \Sigma^*$ such that $x = wyz$ for some $w, z \in \Sigma^*$. If $w \neq \varepsilon$ or $z \neq \varepsilon$, then y is said to be a *proper factor*. If $z = \varepsilon$, then $x = wy$ and y is said to be a *suffix* of x. Similarly, we say that $y \in \Sigma^*$ is a *prefix* of $x \in \Sigma^*$ when $w = \varepsilon$.

If $w = a_1a_2 \cdots a_n$ is a word, then by w^R we mean its *reversal*, that is, $w^R = a_na_{n-1} \cdots a_2a_1$. For example, $(drawer)^R = reward$. Note that $(wx)^R = x^Rw^R$. A word w is a *palindrome* if $w = w^R$.

Given a word $w \in \Sigma^*$, the *projection* of w over an alphabet $\Sigma_0 \subseteq \Sigma$, is the word $P_{\Sigma_0}(w)$ obtained by removing from w all the symbols which are not in Σ_0. For example, if $w = aabbcc$ over $\Sigma = \{a, b, c\}$, then $P_{\{a,b\}}(w)$ is the word $aabb$.

If $x = a_1a_2 \cdots a_n$ and $y = b_1b_2 \cdots b_n$ are two words of the same length, then the *perfect shuffle* of x and y is the word $a_1b_1a_2b_2 \cdots a_nb_n$. For example, The perfect shuffle of *term* and *hoes* is *theorems*.

Now we turn to sets of strings. If Σ is an alphabet, and $L \subseteq \Sigma^*$, then L is a *language over* Σ. A language L is *unary* if $L \subseteq \{a\}^*$ for some letter a, otherwise it is said to be *nonunary*.

Given a language L, we define its *prefix* and *suffix closures* as

$$\mathrm{Pref}(L) = \{x \in \Sigma^* \mid \text{there exists } y \in L \text{ such that } x \text{ is a prefix of } y\}\,;$$

$$\mathrm{Suff}(L) = \{x \in \Sigma^* \mid \text{there exists } y \in L \text{ such that } x \text{ is a suffix of } y\}\,.$$

Now let us introduce some useful operations on languages.

Standard set operations such as union, intersection, and complement can be applied to languages as usual:

- Let L and M be languages over alphabet Σ. Then $L \cup M$ is the language that contains all strings that are in either or both of L and M.

- Let L and M be languages over alphabet Σ. $L \cap M$ is the language that contains all strings that are in both L and M.

- Given an alphabet Σ, the complement of a language $L \subseteq \Sigma^*$ is the language $L^c = \Sigma^* \setminus L$.

The operation of *product* or *concatenation* between two languages L_1 and L_2 is defined by

$$L_1L_2 = \{uv \mid u \in L_1, v \in L_2\}\,.$$

For example, if $L_1 = \{a, b, \ldots, h\}$ and $L_2 = \{1, 2, \ldots, 8\}$, then L_1L_2 represents all chess boards coordinates in algebraic notation.

The *shuffle* of two words w_1 and w_2, not necessarily of the same length, is the set of all words obtained by interleaving the letters as in shuffling a deck of cards. Note this is not the same as the perfect shuffle. We denote it by $w_1 \sqcup\!\sqcup\, w_2$. For instance, $ab \sqcup\!\sqcup\, cd = \{abcd, acbd, acdb, cabd, cadb, cdab\}$ and $ab \sqcup\!\sqcup\, c = \{cab, acb, abc\}$. Formally,

$$\begin{aligned} w_1 \sqcup\!\sqcup\, w_2 = \{u_1v_1 \cdots u_mv_m \mid \forall i = 1, \ldots, m, m \in \mathbb{N},\\ u_i, v_i \in \Sigma^*, w_1 = u_1 \cdots u_m, w_2 = v_1 \cdots v_m\}\,. \end{aligned}$$

The *shuffle* of two languages L_1 and L_2 is defined to be

$$L_1 \sqcup\!\sqcup\, L_2 = \{x \sqcup\!\sqcup\, y \mid x \in L_1, y \in L_2\}\,.$$

We define $L^0 = \{\varepsilon\}$ and $L^k = LL^{k-1}$ for $k \geq 1$. Then the *Kleene closure* or *Kleene star* is the set

$$L^* = \bigcup_{k=0}^{\infty} L^k\,.$$

The *positive closure* of $L \subseteq \Sigma^*$ is the set

$$L^+ = \bigcup_{k=1}^{\infty} L^k\,.$$

Note that L^+ contains ε if and only if L does and in this case $L^+ = L^*$.

If L is a language, then the *reversed language* is defined as

$$L^R = \{x^R \mid x \in L\}\,.$$

We can extend the notion of projection to languages in a standard way. Given a language $L \subseteq \Sigma^*$, the *projection* of L over an alphabet $\Sigma_0 \subseteq \Sigma$ is

$$P_{\Sigma_0}(L) = \{P_{\Sigma_0}(w) \mid w \in L\}\,.$$

For example, if the language $L = \{a^nb^nc^n \mid n \geq 0\}$ over $\Sigma = \{a, b, c\}$, then $P_{\{a,b\}}(L) = \{a^nb^n \mid n \geq 0\}$.

Let $L \subseteq \Sigma^+$. Then a sequence $(x_1, x_2, \ldots, x_n)$ of n words of L is an *L-factorization* of a word $w \in \Sigma^*$ if $w = x_1x_2 \cdots x_n$ [Lot02, Chapter 6].

A set $L \subseteq \Sigma^+$ is a *code* if any word $w \in \Sigma^*$ has at most one L-factorization. For example, the language $L = \{a, ab, ba\}$ over $\Sigma = \{a, b\}$ is not a code because $w = aba$ has two distinct L-factorizations, namely (a, ba) and (ab, a).

The simplest codes are prefix codes. A *prefix code* $L \subseteq \Sigma^+$ is a set such that no word of L is a proper prefix of another word of L. Prefix codes are also known as *prefix-free codes*. For example, the language $L = \{a, ba, bb\}$ over the alphabet $\{a, b\}$ is a prefix-free code. *Suffix codes* are defined symmetrically as sets such that no word of L is a proper suffix of another word of L.

The advantage of codes is that we can decode a given encoded string uniquely.

We now turn to a common notation for representing some kinds of languages. Regular expressions over Σ can be obtained from $\emptyset$, the empty language, $\{\varepsilon\}$, the language consisting of only the empty string and $\{a\}$, $\forall a \in \Sigma$, by a finite number of applications of union, concatenation, and Kleene star. Actually, we can observe that the empty string ε is redundant because it can be obtained by $\emptyset^*$. In other words, a regular expression is a well-formed string over the larger alphabet $\Sigma \cup A$, where $A = \{\varepsilon, \emptyset, (,), +, \cdot, *\}$, assuming $\Sigma \cap A = \emptyset$. In evaluating such an expression, $*$ represents Kleene closure and has the highest precedence. Concatenation is represented by $\cdot$, but sometimes the symbol is omitted, and has next highest precedence. Finally, $+$ represents union and has lowest precedence. Parentheses are used for grouping.

If the word u is a regular expression, then $L(u)$ represents the language that u is shorthand for. For instance, consider the regular expression $u = (a + b)^*a(a + b)(a + b)$. Then $L(u)$ represents all finite words of as and bs where the third symbol from the right is a. A language L is said to be *regular* if $L = L(u)$ for some regular expression u.

The number of languages that we can build over an alphabet is infinite. If the language contains only a finite number of strings, it is easy to represent it: one simply lists the finite set of strings. On the other hand, if the language is infinite, we might not be able to provide any finite representation for the language. Anyway, there exist classes of languages that have a finite representation. Two methods to represent them are through the generative and recognition point of views. The generative method is by grammars that allow to generate all words of the language. The recognition method is through a machine that recognizes the words of the language. In the next sections we will recall these notions.

2.3 Chomsky's hierarchy of grammars

A *grammar* is a tuple $G = (V, \Sigma, P, S)$, where V, Σ, P, and S are, respectively, the *variables*, *terminals*, *productions or rules*, and *start symbol*. The sets V, Σ, and P are finite sets. We assume that V and Σ contain no elements in common, that is, $V \cap \Sigma = \emptyset$. The set of productions P consists of expressions of the form $\alpha \to \beta$, where α is a string in $(V \cup \Sigma)^+$ and β is a string in $(V \cup \Sigma)^*$ [Cho56]. Some authors require that α contains at least one variable [Har78]. Anyway, this does not change the generative power. Finally, S is always a symbol in V.

We have presented a grammar, but have not yet defined the language it generates. To do this, we need to introduce the relations $\Longrightarrow$ and $\overset{\star}{\Longrightarrow}$ between strings in $(V \cup \Sigma)^*$.

We define as *derivation in a single step* the relation $\Longrightarrow$. Given $x, y \in (V \cup \Sigma)^*$, $x \Longrightarrow y$ if there exist $\gamma, \delta \in (V \cup \Sigma)^*$ and $\alpha \to \beta \in P$ such that $x = \gamma\alpha\delta$ and $y = \gamma\beta\delta$.

For each $k \geq 0$, we indicate by $\overset{k}{\Longrightarrow}$ a *derivation in k steps.* Given $x, y \in (V \cup \Sigma)^*$, $x \overset{k}{\Longrightarrow} y$ if there exist $x_0, x_1, \ldots, x_k \in (V \cup \Sigma)^*$ such that $x = x_0, y = x_k$ and $x_{i-1} \Longrightarrow x_i$ for each $i = 1, \ldots, k$. Hence, $x \overset{0}{\Longrightarrow} y$ if and only if $x = y$.

We denote by $\overset{+}{\Longrightarrow}$ a *derivation in one or more steps.* Given $x, y \in (V \cup \Sigma)^*$, $x \overset{+}{\Longrightarrow} y$ if there exists $k \geq 1$ such that $x \overset{k}{\Longrightarrow} y$. The relation $\overset{+}{\Longrightarrow}$ is the transitive closure of $\Longrightarrow$.

Finally, the relation $\overset{\star}{\Longrightarrow}$ denotes a *derivation in an arbitrary number of steps.* Given $x, y \in (V \cup \Sigma)^*$, y is derived from x in an arbitrary number of steps if there exists $k \geq 0$ such that $x \overset{k}{\Longrightarrow} y$. The relation $\overset{\star}{\Longrightarrow}$ is the reflexive, transitive closure of $\Longrightarrow$.

A string of terminals and variables α is called a *sentential form.* We can go from one sentential form to another by applying a rule of the grammar. Hence, a *derivation* consists of zero or more applications of $\Longrightarrow$ to some sentential form.

The *language generated* by G, denoted $L(G)$, is the set of all words in Σ^* that can be derived from S, i.e.,

$$L(G) = \{w \in \Sigma^* \mid S \overset{\star}{\Longrightarrow} w\}\,.$$

When we deal with several grammars, to specify that a derivation is referred to a grammar G, we use $\underset{G}{\overset{\star}{\Longrightarrow}}$ for $\overset{\star}{\Longrightarrow}$ and $\underset{G}{\overset{+}{\Longrightarrow}}$ for $\overset{+}{\Longrightarrow}$.

We say that two grammars G_1 and G_2 are *equivalent* if $L(G_1) = L(G_2)$.

The grammars are classified according to the type of production. The more general grammars, which have no restriction on production, are called *type* 0 *grammars.* The class of languages generated by these grammars coincides with the class of languages recognized by Turing machines and it is often called the *recursively enumerable languages.*

Let $G = (V, \Sigma, P, S)$ be a grammar. Suppose that for every production $\alpha \to \beta \in P$, $|\alpha| \leq |\beta|$. Then the grammar G is *type* 1 or *context sensitive* (csg). The class of languages generated by these grammars coincides with the class of languages recognized by linear bounded automata.

Now, suppose that every production in P is of the form $A \to \beta \in P$, where $A \in V$ and $\beta \in (V \cup \Sigma)^+$. Then the grammar is called *type* 2 or *context free* (cfg). It can be noted that a production of the form $A \to \beta$ allows the variable A to be replaced by the string β independently of the context in which A appears, hence named context free. The class of languages generated by these grammars coincides with the class of languages recognized by pushdown automata.

Further, suppose that every production in P is of the form $A \to aB$ or $A \to a$, where A and B are variables and a is a terminal. Then the grammar is called a *type* 3 or *regular grammar* (rg). The class of languages generated by these grammars coincides with the class of languages recognized by finite state automata. [1]

It should be clear that every regular grammar is context free; every context-free grammar is context sensitive; every context-sensitive grammar is of type 0.

We shall call a language that can be generated by a type 0 grammar a *type* 0 *language*. A language generated by a context-sensitive, context-free, or regular grammar is a *context-sensitive* (csl), *context-free* (cfl), or *regular* (rl) *language*, respectively.

Figure 2.1 shows the classes of languages according to Chomsky [Cho56, Cho59]. This thesis deals with regular and context-free languages.

We summarize the *closure properties* for regular and context-free languages in Table 2.1. A detailed investigation on these closure properties are presented in [HMU01, Sections 4.2,7.3] and [CSY02, JM12]. We see that some, but not all, of the closure properties that the rls have are also possessed by the cfls.

The context-free languages are closed under the following operations: union, concatenation, star, reversal and projection. For the closure property under projection we could proceed in the following way: if $A \to \beta$ is a production of the grammar defined over Σ, then $P_{\Sigma_0}(L)$ where $\Sigma_0 \subseteq \Sigma$ can be obtained by removing from β all terminals which are not in Σ_0. Unlike the regular languages, the cfls are not closed under intersection, complement or shuffle. For example, the shuffle of cfls $L_1 = \{a^n b^n \mid n \geq 0\}$ and $L_2 = \{c^m d^m \mid m \geq 0\}$ is not a cfl.

[1] We might note that, the definition excludes ε from context-sensitive, context-free or regular languages. We shall extend our definition of csg, cfg, and rg to allow productions of the form $S \to \varepsilon$, where S is the start symbol, provided that S does not appear on the right-hand side of any production. In this case, it is clear that the production $S \to \varepsilon$ can only be used as the first step in a derivation. We can observe that if L is a context-sensitive, context-free, or regular language, then $L \cup \{\varepsilon\}$ and $L \setminus \{\varepsilon\}$ are csls, cfls, or rls, respectively.

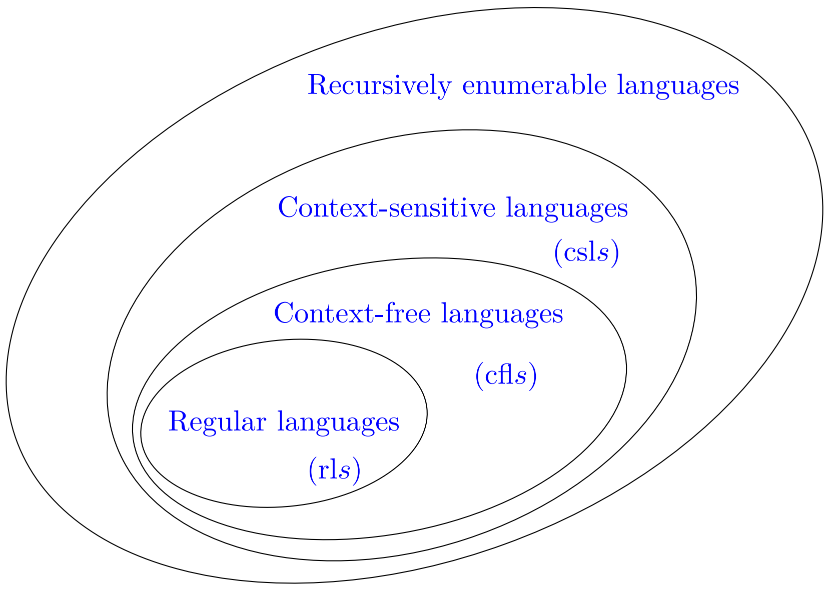

Figure 2.1: Chomsky's hierarchy of languages.

2.4 Context-free grammars

A context-free grammar is said to be in *Chomsky normal form* if all its productions are in one of the three simple forms, either $B \to CD$, $B \to a$, or $S \to \varepsilon$, where $a \in \Sigma$, $B \in V$, and $C, D \in V \setminus \{S\}$. The cfgs in Chomsky normal form are called *Chomsky normal form grammars* (Cnfgs).

Theorem 2.4.1 (Chomsky Normal Form)**.** *Every context-free language is generated by a grammar in Chomsky normal form.*

A useful way of representing the derivations in a context-free grammar is by parse trees. These trees show us clearly how the symbols of a terminal string are grouped into substring, each of which belongs to the language of one of the variables of the grammar. Let us consider a cfg $G = (V, \Sigma, P, S)$. A *(parse or derivation) tree* T for G is a tree satisfying the following conditions:

- Each interior node is labeled by a variable in V.
- The root is labeled by a variable in V.
- Each leaf is labeled by either a variable, a terminal, or ε. However, if the leaf is labeled ε, then it must be the only child of its parent S.

Closed under	Regular languages	Context-free languages
Union	Yes	Yes
Intersection	Yes	No
Complement	Yes	No
Concatenation	Yes	Yes
Star	Yes	Yes
Shuffle	Yes	No
Reversal	Yes	Yes
Projection	Yes	Yes

Table 2.1: Closure properties of the classes of regular and context-free languages.

- If an interior node is labeled with $A \in V$, and its children are labeled

  $$X_1, X_2, \ldots, X_k \in (V \cup \Sigma)$$

  respectively, from the left, then $A \to X_1 X_2 \cdots X_k$ is a production in P. Note that the only chance that one of the X's can be ε is if this is the label of the single child, and $A \to \varepsilon$ is a production of G.

If we look at the leaves of any parse tree T and concatenate them from the left, we get a string $x \in (V \cup \Sigma)^*$, called the *yield* of the tree, which is always a string that is derived from a root variable A. In this case we write $T : A \overset{\star}{\Longrightarrow} x$.

Of special importance are those parse trees such that the root is labeled by the start symbol and the yield is a terminal string, that is, all leaves are labeled either with a terminal or with ε. These are parse trees whose yields are strings in the language of the underlying grammar.

Let us introduce the concept of *leftmost derivation*, that is a derivation in which the variable replaced at each step is the leftmost one. A grammar G is said to be *unambiguous* if every word $w \in L(G)$ has exactly one leftmost derivation and *ambiguous* otherwise.

We can construct a (leftmost) derivation from a parse tree and vice versa. The matter of ambiguity in grammars and languages is an important application of parse trees. A grammar is unambiguous, if every word generated has exactly one parse tree.

2.5 Finite state automata

In this section we see another method of finitely specifying infinite languages, the recognizer. We consider what is undoubtedly a very simple recognizer, called a *finite automaton.* The languages defined are exactly the regular languages.

We may visualize a finite state automaton as in Figure 2.2. The model consists of a *finite control*, which reads symbols from a linear input tape. The *tape* is divided into squares of *cells*; each cell can hold any of a finite number of symbols. There is a *tape head* that is always positioned at one of the tape cells. We say that the machine is *scanning* that cell.

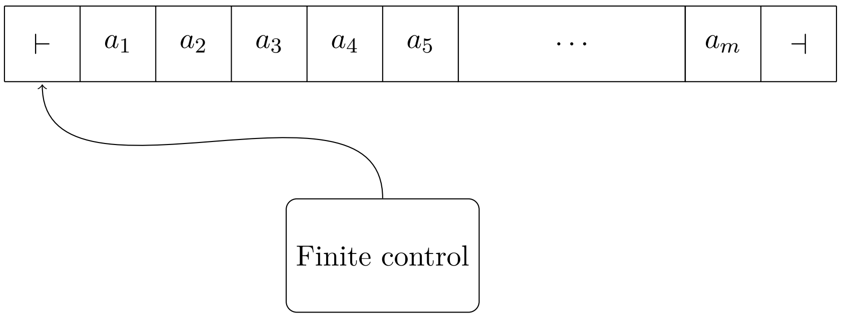

Figure 2.2: A finite state automaton.

Initially, the *input string*, which is a finite-length string of symbols, chosen from the *input alphabet*, is placed on the tape. The input string is enclosed in *left* $\vdash$ and *right* $\dashv$ *endmarkers*, which are not elements of the input alphabet.

Informally, the finite control starts in its *initial state* with its reading head pointing to the left endmarker. At any point in time, the machine is in some state with its reading head scanning some tape cell containing an input symbol or one of the endmarkers. Based on its current state and the symbol occupying the tape cell, it *moves* its reading head one position leftward, one position rightward or remains on the same cell. The reading head may not move outside of the endmarkers. The machine move on a particular state and symbol is determined by a *transition function*, which is part of the specification of the machine.

The finite state automata are classified according to the transition function. While in *one-way automata* the input string is scanned from left to right, until reaching the end of the input, where the word is accepted or rejected, in *two-way automata* the head can be moved in both directions. Moreover, unlike two-way automata, one-way automata do not need the endmarkers. Now we introduce the formal definitions of finite state automata.

Definition 2.5.1. *A* two-way nondeterministic finite automaton *(*2nfa*) is defined as a 7-tuple $A = (Q, \Sigma, \vdash, \dashv, \delta, q_0, F)$ in which:*

- Q *is the finite set of* states*.*
- Σ *is the finite* input alphabet*.*
- $\vdash, \dashv \notin \Sigma$ *are two special symbols, called the* left *and the* right endmarkers*, respectively.*
- $\delta : Q \times (\Sigma \cup \{\vdash, \dashv\}) \to 2^{Q\times\{-1,0,+1\}}$ *is the* transition function*.*
- $q_0 \in Q$ *is the* initial state*.*
- $F \subseteq Q$ *is the set of* accepting (final) states*.*

The input string is stored on the tape surrounded by the two endmarkers. The cells of the tape are numbered from left to right, beginning with zero for the left endmarker. In one move, the automaton A reads an input symbol, changes its state, and moves the reading head one cell to the right, left, or keeps it *stationary*, depending on whether δ return $+1, -1$ or 0, respectively.

If, for some $q \in Q$ and $a \in \Sigma \cup \{\vdash, \dashv\}$, we have $|\delta(q, a)| > 1$, the machine makes a *nondeterministic choice*. If $|\delta(q, a)| = 0$, the machine *halts* and the state is said to be *halting*.

The machine *accepts* the input string, if there exists a computation path from the initial state q_0 with head on the left endmarker to the right endmarker into some accepting state $q \in F$.

The *language accepted* or *defined* by an automaton A, denoted $L(A)$, consists of all input strings that are accepted by the automaton A.

Definition 2.5.2. *A two-way nondeterministic automaton* $A = (Q, \Sigma, \vdash, \dashv, \delta, q_0, F)$ *is said to be* two-way deterministic *(*2dfa*), if* $|\delta(q, a)| \leq 1$ *for all* $q \in Q$ *and* $a \in \Sigma \cup \{\vdash, \dashv\}$*.*

We assume that a 2dfa starts the computation in a designed initial state, scanning the left endmarker and that its head cannot violate the endmarkers, namely, there are no transitions reading the left (right) endmarker and moving to the left (right, respectively). In the literature, several slightly different acceptance conditions for two-way automata have been considered [RS59, SS78, KMP14, KP12]. Here, we assume that a 2dfa accepts the input by entering in a special unique state q_f which is also halting.

Now we introduce one-way automata.

Definition 2.5.3. *A* one-way nondeterministic finite automaton *(*1nfa*) is defined as a* 5*-tuple* $A = (Q, \Sigma, \delta, q_0, F)$ *where:*

- Q *is the finite set of* states*.*
- Σ *is the finite* input alphabet*.*
- $\delta : Q \times \Sigma \to 2^Q$ *is the* transition function*.*

- $q_0 \in Q$ *is the* initial state*.*
- $F \subseteq Q$ *is the set of* accepting (final) states*.*

We need to extend the transition function δ to a function $\widehat{\delta}$ that takes a state q and an input string w, and returns the set of states the automaton is in, if it starts in state q and processes the string w. The transition function $\widehat{\delta} : Q \times \Sigma^* \to 2^Q$ is defined as:

- $\forall q \in Q, \widehat{\delta}(q, \varepsilon) = q$
- $\forall x \in \Sigma^*, \forall a \in \Sigma, \widehat{\delta}(q, xa) = \bigcup_{p \in \widehat{\delta}(q,x)} \delta(p, a)$

In the following we will use just the symbol δ also for $\widehat{\delta}$.

The machine *accepts* the input, if there exists a computation path from the initial state q_0 to some accepting state $q \in F$. The *language accepted* or *defined* by a 1nfa, denoted $L(A)$, consists of all input strings that are accepted by the automaton A, i.e.,

$$L(A) = \{w \in \Sigma^* \mid \delta(q_0, w) \cap F \neq \emptyset\} \; .$$

Definition 2.5.4. *A one-way nondeterministic automaton $A = (Q, \Sigma, q_0, F)$ is a* one-way deterministic *(*1dfa*), if it has no nondeterministic choices, i.e., if* $\forall q \in Q, \forall a \in \Sigma, |\delta(q, a)| = 1$*.*

A 1dfa is said to be *complete* if $\delta(q, a)$ is always defined. In our terminology, a 1dfa is always complete unless it is explicitly stated to be incomplete.

We call *unary* any automaton that works with a single letter input alphabet.

An automaton is *halting* if no computation path can get into an infinite loop, that is, on every input, each computation path halts after a finite number of steps. Note that infinite computations are possible in two-way automata, while they are not possible in one-way automata. This is because of the move of the reading head from left to right and vice versa.

Two automata A_1 and A_2 are *equivalent* if $L(A_1) = L(A_2)$.

Considering one-way and two-way automata, with their deterministic and nondeterministic versions, we obtain four types of automata which are equivalent in constructive way. That is, a language L is accepted by a 2nfa if and only if L is accepted by a 2dfa if and only if L is accepted by a 1nfa if and only if L is accepted by a 1dfa. Hence, all these variants of finite automata recognize the same class of languages, namely the class of regular languages.

A useful way of representing the finite state automata is by a directed graph, called *state diagram.* This consists of a node for every state and a directed line from state q to state p with label $a \in \Sigma$ if the finite automaton, in state q, scanning the input symbol a, would go to state p. Final states are indicated by a double circle. The initial state is indicated by an unlabeled arrow entering in the state.

Example 2.5.1. *The state diagram in Figure 2.3 represents a one-way nondeterministic automaton that accepts the language L_3, where*

$$L_n = \{x \in \{a,b\}^* \mid \textit{the nth symbol from the right is } a\}\,.$$

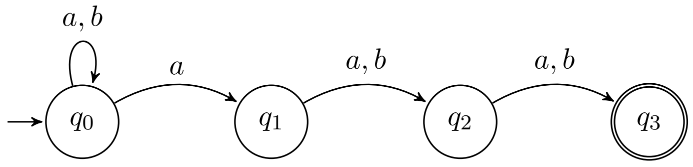

Figure 2.3: State diagram for a 1nfa.

□

The language L_3 accepted by the automaton in Figure 2.3 is representable by the following regular expression $(a+b)^*a(a+b)(a+b)$. The regular expressions, introduced in Section 2.2, are an algebraic notation that describe exactly the same languages as finite automata: the regular languages. The equivalence between regular expressions and finite automata has been shown by Kleene in [Kle56]. There is a correspondence in both directions and this is formalized in Kleene's Theorem.

Theorem 2.5.1. *(Kleene's Theoem) The following language classes are identical:*

- *the class of languages specified by regular expressions.*
- *the class of languages accepted by* 1dfa*s.*
- *the class of languages accepted by* 1nfa*s.*

It is possible to show that the smallest 1dfa accepting L_n has at least 2^n states, so 1nfas, while accepting the same class of languages as 1dfas, can be exponentially more concise. This is the topic of the next chapter.

Chapter 3

Descriptional complexity

Adagio

Clarinet concerto in A Major
Wolfgang Amadeus Mozart

In this chapter, we introduce a few basic notions about descriptional complexity. Moreover, fundamental properties of descriptional systems and their complexity measures are discussed and presented in a unified manner [GKK$^+$02, HK10].

A natural and important measure of descriptional complexity is the size of a representation of a language, that is, the length of its description. This chapter is devoted to several aspects and results with respect to complexity measures that are recursively related to the sizes.

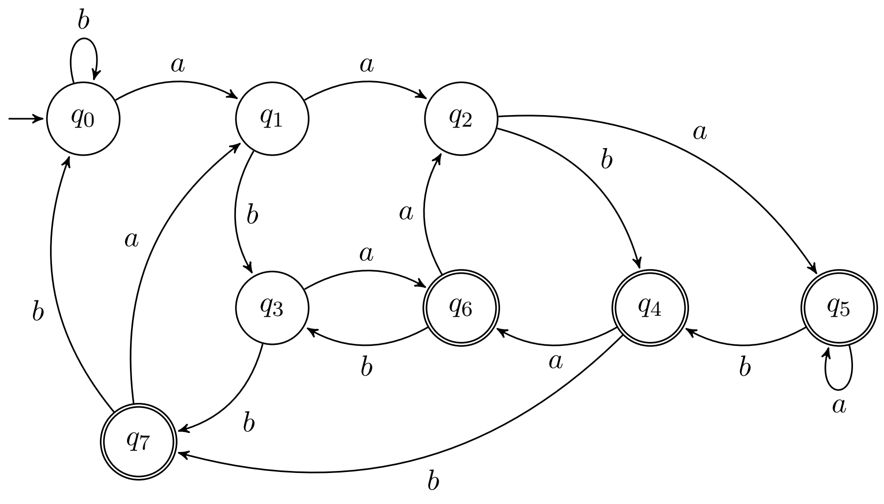

Figure 3.1: A 1dfa equivalent to 1nfa in Figure 2.3.

A comprehensive overview of results is given concerning the question:

> how succinctly can a regular or a context-free language be represented by a descriptor of one descriptional system compared with the representation by an equivalent descriptor of another descriptional system?

To illustrate the situation consider the language $L_3 = (a+b)^*a(a+b)(a+b)$ introduced in Example 2.5.1 where

$$L_n = \{x \in \{a,b\}^* \mid \text{the } n\text{th symbol from the right is } a\}\,.$$

Figure 3.1 shows a 1dfa equivalent to the 1nfa showed in Figure 2.3. The automaton has been obtained by subset construction. Observe that the states unreachable in the automaton are not shown. It is possible to show that the smallest complete 1dfa has exactly 2^n states while there is a 1nfa with $n+1$ states accepting L_n. This gives an exponential gap. However, this example does not give the maximum blow-up. A more sophisticated example, where the maximum blow-up is reached, can be found in [MF71].

This exponential trade-off between one-way nondeterministic and deterministic finite automata is a fundamental result in this area. Using the classical subset construction [RS59], from each 1nfa with n states, we can build an equivalent 1dfa with 2^n states. Moreover, this state bound cannot be reduced in the worst case. That is, it is optimal [Lup63, MF71, Moo71].

Now we first establish some notation for descriptional complexity.

3.1 Descriptional systems

We formalize the intuitive notion of a representation or description of a family of languages. A *descriptional system* is a collection of encodings of items where each item *represents* or *describes* a formal language. In the following, we call the items *descriptors*. Formally,

Definition 3.1.1. *A* descriptional system S *is a set of non-empty finite descriptors, such that each descriptor* $D \in S$ *describes a formal language* $L(D)$.

The family of languages represented (or described) by some descriptional system S is

$$L(S) = \{L(D) \mid D \in S\}\,.$$

For every language L, the set of its descriptors in a descriptional system S is

$$S(L) = \{D \in S \mid L(D) = L\}\,.$$

For instance, 1nfas are a descriptional system S_1 and $L(S_1)$ is the family of regular languages. The family of context-free grammars is other descriptional system S_2 and $L(S_2)$ is the family of context-free languages.

Now we turn to measure the size of descriptors.

3.2 Complexity measure

Basically, we are interested in defining a complexity measure as general as possible to cover a wide range of approaches, and in defining it as precise as necessary to allow a unified framework for proofs.

Common notions as the relative *succinctness of descriptional systems* and our intuitive understanding of descriptional complexity suggest to consider *the size of descriptors*.

For instance, note that for each $n \geq 1$, the regular language $L_n = (a^n)^*$ requires n states to be recognized by a 1nfa or by a 1dfa. Similar considerations can be formulated for grammars. The language L_n is generated by the grammar containing only one variable S and the productions $S \to a^n$ and $S \to a^n S$. Observe that for large n we have always *one* variable and *two* productions to express this language. The number of variables or productions are not a measure for grammars in general, but it becomes a measure if the descriptional system is chosen to be that of grammars in some special form. Another example is treated in [HK10, Example 1.71]. However, it is not difficult to see that for grammars in Chomsky normal form, the number of variables is a “reasonable” measure of complexity [Gru73]. Formally,

Definition 3.2.1. *Given a descriptional system S, a* complexity (or size) measure *for S is a total, recursive function $c : S \to \mathbb{N}$ such that for any alphabet Σ, the set of descriptors in S describing languages over Σ is recursively enumerable in order of increasing size and, moreover, does not contain infinitely many descriptors of the same size.*

For $D \in S$, $c(D)$ is the complexity of the descriptor *D.*

We will call “reasonable measure” a measure with the properties defined above.

In this thesis we consider as complexity measures for context-free grammars the number of variables of the grammar in Chomsky normal form and for finite automata the number of states. Further measures based on other criteria induced by grammatical levels, derivation trees, derivation steps, etc., are introduced and studied in [Gru71, Gru76].

Whenever we consider the relative succinctness of two descriptional systems S_1 and S_2, we assume the intersection $L(S_1) \cap L(S_2)$ to be nonempty.

Definition 3.2.2. *Let S_1 be a descriptional system with complexity measure c_1, and S_2 be a descriptional system with complexity measure c_2. A total function $f : \mathbb{N} \to \mathbb{N}$, is said to be an* upper bound *for the increase in complexity when changing from a descriptor in S_1 to an equivalent descriptor in S_2, if for all $D_1 \in S_1$ with $L(D_1) \in L(S_2)$ there exists a $D_2 \in S_2(L(D_1))$ such that*

$$c_2(D_2) \leq f(c_1(D_1)) .$$

If there is no recursive function serving as upper bound, the trade-off is said to be *non-recursive.*

Definition 3.2.3. *Let S_1 be a descriptional system with complexity measure c_1, and S_2 be a descriptional system with complexity measure c_2. A total function $f : \mathbb{N} \to \mathbb{N}$, is said to be a* lower bound *for the increase in complexity when changing from a descriptor in S_1 to an equivalent descriptor in S_2, if for infinitely many $D_1 \in S_1$ with $L(D_1) \in L(S_2)$ there exists a minimal $D_2 \in S_2(L(D_1))$ such that*

$$c_2(D_2) \geq f(c_1(D_1)) \,.$$

A main field of investigation deals with the question: how succinctly can a language be represented by a descriptor of one descriptional system compared with the representation by an equivalent descriptor of the other descriptional system? An upper bound for the trade-off gives the maximal gain in economy of description, and conversely, the maximal blow-up (in terms of descriptional complexity) for simulations between the descriptional systems. A maximal lower bound for the trade-off determines the costs which are necessary in the worst cases.

3.3 Optimal simulations

Finite automata are one of the most elementary computational models. They are used in many areas of computer science, from process modeling in software engineering to protocol specification in distributed systems. First of all, their computational power is well established: they exactly characterize the class of regular languages. A classic example of descriptional complexity result in this field is the comparison, in terms of states, between 1dfa and 1nfa. It is well known that, for any integer n, 2^n states are necessary [Lup63, MF71, Moo71] and sufficient [RS59] for a 1dfa to simulate n-state 1nfa. In other words, in [Lup63, MF71, Moo71] it is shown that the subset construction cannot be improved.

Rabin and Scott [RS59] and Shepherdson [She59] independently discovered that the capability of moving the input head in both directions does not increase the recognition power of finite memory devices. Namely, also 2dfa or 2nfa characterize regular languages, exactly as their one-way deterministic and nondeterministic counterparts. The fact that all models of finite automata introduced so far have the same computational power encourages to compare their succinctness, i.e., to compare the size of different automata types required to accept the same language.

Figure 3.2 shows the well-known costs, in terms of states, of simulating different n-state automata by 1dfa: one-way nondeterministic finite automata: $O(2^n)$ [RS59], two-way deterministic finite automata: $O(n^n)$ [RS59, She59],

$$\begin{tikzcd}
\text{1nfa} \arrow[r, "?"] \arrow[dr, "O(2^n)"] & \text{2dfa} \arrow[d, "O(2^{n\log n})"] & \text{2nfa} \arrow[l, "?"'] \arrow[dl, "O(2^{n^2})"] \\
& \text{1dfa} &
\end{tikzcd}$$

Figure 3.2: Optimal simulations between automata.

two-way nondeterministic finite automata: $O(2^{n^2})$ [RS59, She59]. All these bounds are tight. More detailed bounds are studied in [Kap05].

Nevertheless, some challenging problems concerning finite automata are still open. An important example is the question posed by Sakoda and Sipser in 1978 [SS78] about the existence of a polynomial simulation of 2nfa or 1nfa by 2dfa. They conjecture that these simulations are exponential. Several authors treated this problem, obtaining some results for restricted models, e.g, sweeping automata [Sip80, Ber80, Mic81], oblivious automata [HS03], unary automata [Chr86, MP01, GMP03]. However, a solution to the general problem seems to be very far.

A sweeping automaton is a 2dfa that changes direction only at either end of the input. Sipser [Sip80] showed that there exists a family of n-state 1nfa over an alphabet of size 2^{n^2} such that any equivalent sweeping automaton requires at least 2^n states. In [Ber80] and [Mic81] it is shown that 2dfa can be exponentially more succinct than sweeping automata. The result of Sipser was generalized by Hromkovic and G. Schnitger [HS03] considering oblivious machines (moving the input head along the same trajectory on all inputs of the same length).

Berman and Lingas [BL77] state a polynomial lower bound of $\Omega(n^2/\log n)$ for cost of 2nfa by 2dfa simulation, and provide an interesting connection with the celebrated open problem $\mathrm{L} \overset{?}{=} \mathrm{NL}$ (where L and NL are the classes of languages accepted in logarithmic space by deterministic and nondeterministic Turing machines, respectively). More precisely, they show that if $\mathrm{L} = \mathrm{NL}$, then for some polynomial p, for all integers m, and all k-state 2nfa A, there is a $p(mk)$-state 2dfa accepting the set of all strings of length at most m in $L(A)$. As a consequence of this result, Sipser [Sip80] relates the $\mathrm{L} \overset{?}{=} \mathrm{NL}$ question also to the existence of sweeping automata with a polynomial number of states for a certain family of regular languages. A further improvement is contained in [Chr86], where the best known lower bound is $\Omega(n^2)$ for the cost of 2nfas by 2dfas simulation. As the L vs. NL problem is one of the major open problems in the structural complexity theory, it gives additional motivations for the study of the relationship between 2dfas and 2nfas.

One of the most promising restrictions of the open question of Sakoda and Sipser is represented by its unary version which leads us to study optimal simulations between unary automata. The following are the main results on descriptional complexity of finite automata over unary alphabets:

- Each n-state 1nfa can be converted to a $O(n^2)$-state 2dfa, and the conversion is tight [Chr86].
- Each n-state 1nfa can be converted to a $e^{O(\sqrt{n\cdot\ln n})}$-state 1dfa [Chr86].
- Each n-state 2dfa can be converted to a $e^{O(\sqrt{n\cdot\ln n})}$-state 1dfa (and 1nfa) [Chr86].
- Each n-state 2nfa can be converted to a $e^{O(\sqrt{n\cdot\ln n})}$-state 1dfa [MP01].

They are shown in Figure 3.3.

$$\begin{tikzcd}
\text{1nfa} \arrow[r, bend left, "n^2"] \arrow[dr, "e^{\sqrt{n\cdot\ln n}}"] & \text{2dfa} \arrow[l, "e^{\sqrt{n\cdot\ln n}}"] \arrow[d, "e^{\sqrt{n\cdot\ln n}}"] & \text{2nfa} \arrow[l, "?"] \arrow[dl, "e^{\sqrt{n\cdot\ln n}}"] \\
& \text{1dfa} &
\end{tikzcd}$$

Figure 3.3: Optimal simulations between different kinds of unary automata.

In [GMP03] the authors proved that each n-state unary 2nfa can be simulated by an equivalent 2dfa with at most $O(n^{\lceil \log_2 (n+1)+3 \rceil}) = 2^{O((\log_2 n)^2)}$ states, hence obtaining a subexponential but still superpolynomial upper bound. Up to now, it is not known whether or not that simulation is tight.

However, a positive answer to this question would imply the separation between the classes L and NL. In fact, in [GP11] it was shown that if L = NL then each unary 2nfa with n states can be simulated by a 2dfa with a number of states polynomial in n. In [GMP07] the problem of the complementation of unary 2nfa was considered. The authors proved that for each n-state 2nfa accepting a unary language L there exists a 2nfa with $O(n^8)$ states accepting the complement of L. They also discuss the relationships between the problem of the complementation of 2nfas and the problem of Sakoda and Sipser.

The studies shown above give evidence that research in descriptional complexity of automata is not only motivated by the investigation of succinct representation of regular languages, but are also linked to fundamental questions in computational and structural complexity.

3.4 Preliminary constructions

Here, we present some preliminary constructions which will be used in the rest of the thesis. These constructions are simple and standard.

Throughout the section, let us fix an alphabet $\Sigma = \{a_1, a_2, \ldots, a_m\}$ of m symbols. Given a language $L \subseteq \Sigma^*$, the *unary parts* of L are the languages

$$L_1 = L \cap \{a_1\}^*,\ L_2 = L \cap \{a_2\}^*, \ldots,\ L_m = L \cap \{a_m\}^*,$$

while the *nonunary part* is the language

$$L_0 = L \setminus (L_1 \cup L_2 \cup \ldots \cup L_m),$$

i.e., the set of all nonunary words belonging to the language L. Clearly,

$$L = \bigcup_{i=0}^{m} L_i$$

First, we consider some *decomposition results*: we show how to obtain automata and grammars for the unary and nonunary parts of the languages defined by given automata and grammars, respectively. After, we will shortly discuss some *composition results*: how to obtain 1dfas or 2dfas, respectively, accepting the union of languages defined by given 1dfas or 2dfas.

3.4.1 Decomposition results

Let us start by considering finite automata. Given an automaton A, we can readily build automata accepting the unary and nonunary parts of $L(A)$, respectively.

Lemma 3.4.1. *For each n-state* 1nfa *A over an m-letter alphabet, there exist $m+1$* 1nfa*s $A_0, A_1, \ldots, A_m$ such that:*

- *A_0 has $n(m+1)+1$ states and accepts the nonunary part of $L(A)$.*
- *For $i = 1, \dots, m$, A_i is a unary* 1nfa *of n states which accepts the unary part $L(A) \cap \{a_i\}^*$.*

Furthermore, if A is deterministic then so are $A_0, A_1, \dots, A_m$.

Proof. To accept an input w, the automaton A_0 has to check that w is accepted by A and contains at least two different symbols. To do that, A_0 uses the same states and transitions as A. However, in a preliminary phase, it keeps track in its finite control of the first letter of w, until discovering a different letter.

The automaton A_0, besides all states and transitions of A, has a new initial state and m extra copies $[q,1], \dots, [q,m]$ of each state q of A. The transitions from the initial state of A_0 simulate those from the initial state of A, also remembering the first symbol of the input, i.e., a transition in A which from the initial state, reading a symbol a_i, leads to the state q, is simulated in A_0 by a transition leading to $[q,i]$.

From a state $[q,i]$, reading the same symbol a_i the automaton A_0 can move to each state $[p,i]$ such that A from q reading a_i can move to p. In this way, until the scanned input prefix consists only of occurrences of the same letter a_i, A_0 simulates A using the ith copies of the states. However, when in a state $[q,i]$ a symbol $a_j \neq a_i$ is read, having verified that the input contains at least two different letters, A_0 can move to each state p which is reachable in A from the state q, so entering the part of A_0 corresponding to the original A. The final states of A_0 are the final states in the copy of A. From this construction we see that the number of states of A_0 is $n(m+1)+1$. Furthermore, if A is deterministic then also A_0 is deterministic.

For the unary parts, it is easily seen that for $i = 1, \dots, m$, the automaton A_i can be obtained by removing from A all the transitions on the symbols $a_j \neq a_i$. Clearly, also this construction preserves determinism. □

Let us see with an example how the lemma proceeds.

Example 3.4.1. *Let us consider a* 1dfa *A as in Figure 3.4 accepting the language*

$$L(A) = \{w \in \{a_1, a_2\}^* \mid |w|_{a_2} \bmod 2 = 0\}.$$

The unary and nonunary parts of the automaton are shown in Figure 3.5 and Figure 3.6, respectively.

□

We can give a similar result in the case of cfgs. The results concerning context-free grammars rely on the number of variables as complexity measure. Since this does not make sense for general grammars, we consider grammars

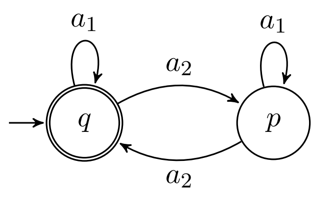

Figure 3.4: A 1dfa A of Example 3.4.1.

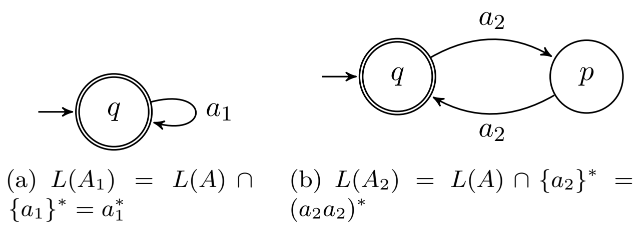
q
a_1
(a) $L(A_1) = L(A) \cap \{a_1\}^* = a_1^*$
a_2
q
p
a_2
(b) $L(A_2) = L(A) \cap \{a_2\}^* = (a_2a_2)^*$

Figure 3.5: Unary parts of the automaton A in Figure 3.4.

in Chomsky normal form. In order to make the results more comparable, we report a known result about the size blow-up for converting a general context-free grammar into Chomsky normal form [HMU01, Sections 7.1,7.4.2]:

Theorem 3.4.1. *Given a* cfg *G having a description of length h, we can find an equivalent* Cnfg *for G in time $O(h^2)$. The resulting grammar has length $O(h^2)$.*

Lemma 3.4.2. *For each h-variable* Cnfg *G generating a language $L(G) \subseteq \Sigma^*$, there exist $m+1$* Cnfg*s $G_0, G_1, \ldots, G_m$ such that:*

- *G_0 has $mh - m + 1$ variables and generates the nonunary part L_0 of $L(G)$.*
- *For $i = 1, \ldots, m$, G_i is a unary* Cnfg *with h variables which generates the unary part $L_i = L(G) \cap \{a_i\}^*$.*

Proof. For $i = 1, \ldots, m$, the design of G_i is simply done by deleting from P all productions of the form $B \to a_j$ with $i \neq j$. Built in this manner, it is impossible for G_i to contain more than h variables.

Giving a linear upper bound on the number of variables for G_0 is slightly more involved. The construction is based on the fact that if any nonunary word w is split into two parts at any position, that is, $w = uv$, then there must exist two different letters $a_i \neq a_j$ such that a_i is contained in u and a_j is contained in v. Actually, a_i, a_j depend not only on w, but also on the position of the boundary between u and v. It is clear that any production

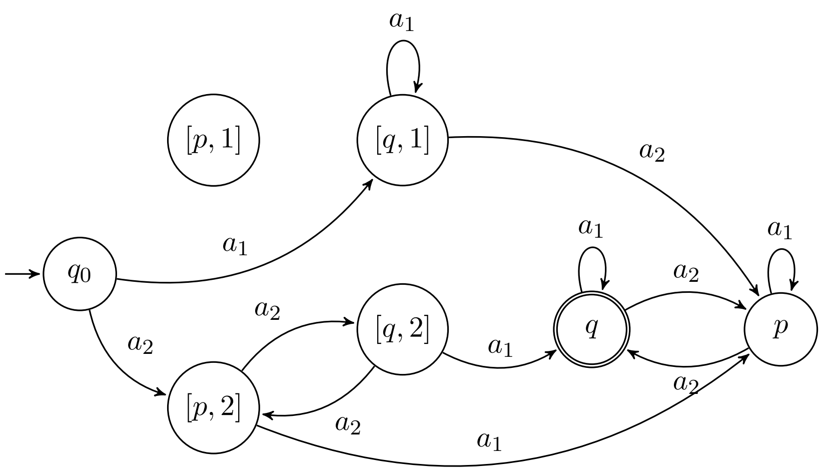

Figure 3.6: Nonunary part of the automaton A in Figure 3.4.

of one letter or ε directly from S is contrary to the purpose of G_0. This observation enables us to focus on the derivations by G that begin with replacing S by two variables. Consider a derivation

$$S \underset{G}{\Longrightarrow} BC \overset{+}{\underset{G}{\Longrightarrow}} uC \overset{+}{\underset{G}{\Longrightarrow}} uv$$

for some nonempty words $u, v \in \Sigma^+$ and $S \to BC \in P$. G_0 simulates G, but also requires extra feature to test whether u and v contain respectively letters a_i and a_j, for some $i \neq j$, and make only derivations that pass this test valid. To this end, we let the start variable S' of G_0 make guess which of the two distinct letters in Σ have to derive from B and C, respectively. We encode this guess into the variables in $V \setminus \{S\}$ as a subscript like B_i (this means that, for $w \in \Sigma^*$, $B_i \overset{+}{\underset{G_0}{\Longrightarrow}} w$ if and only if $B \overset{+}{\underset{G}{\Longrightarrow}} w$ and w contains at least one a_i).

Now, we give a formal definition of G_0 as a quadruple (V', Σ, P', S'), where

$$V' = \{S'\} \cup \{B_i \mid B \in V \setminus \{S\}, 1 \leq i \leq m\}$$

and P' consists of the following production rules:

1. $\{S' \to B_iC_j \mid S \to BC \in P$ and $1 \leq i, j \leq m$ with $i \neq j\}$;
2. $\{B_i \to C_iD_j, B_i \to C_jD_i \mid B \to CD \in P$, $B \neq S$ and $1 \leq i, j \leq m\}$;
3. $\{B_i \to a_i \mid B \to a_i \in P$ and $1 \leq i \leq m\}$.

We conclude the proof by checking that $L(G_0) = \{w \in L(G) \mid w$ is nonunary$\}$. To this aim we prove the following:

Claim. *Let B_i be a variable of G_0 that is different from the start variable. For $w \in \Sigma^*$, $B_i \xRightarrow[G_0]{+} w$ if and only if $B \xRightarrow[G]{+} w$ and w contains at least one occurrence of a_i.*

Both implications will be proved using induction on the length of derivations.

(Only-if part): If $B_i \xRightarrow[G_0]{} w$ (single-step derivation), then w must be a_i and $B \to a_i \in P$ according to the 3rd item in the definition of productions in P'. Hence, the base case is correct. The longer derivations must begin with either $B_i \to C_i D_j$ or $B_i \to C_j D_i$ for some $B \to CD \in P$ and some $1 \le j \le m$. It is enough to investigate the former case (the other one is completely similar). Then we have

$$B_i \xRightarrow[G_0]{} C_i D_j \xRightarrow[G_0]{+} w_1 D_j \xRightarrow[G_0]{+} w_1 w_2 = w$$

for some $w_1, w_2 \in \Sigma^+$. By induction hypothesis, $C \xRightarrow[G]{+} w_1$, w_1 contains a_i, and $D \xRightarrow[G]{+} w_2$. Hence, $B \xRightarrow[G]{} CD \xRightarrow[G]{+} w_1 D \xRightarrow[G]{+} w_1 w_2 = w$ is a valid derivation by G, and w contains a_i.

(If part): The base case is proved as for the direct implication. If $B \xRightarrow[G]{+} w$ is not a single-step derivation, then it must start with applying to B some production $B \to CD \in P$. Namely,

$$B \xRightarrow[G]{} CD \xRightarrow[G]{+} w'_1 D \xRightarrow[G]{+} w'_1 w'_2 = w$$

for some nonempty words $w'_1, w'_2 \in \Sigma^+$. Thus, either w'_1 or w'_2 contains a_i; let us say w'_1 does (the other case is similar). By induction hypothesis, $C_i \xRightarrow[G_0]{+} w'_1$. A letter a_j occurring in w'_2 is chosen, and the hypothesis gives $D_j \xRightarrow[G_0]{+} w'_2$. As a result, the derivation

$$B_i \xRightarrow[G_0]{} C_i D_j \xRightarrow[G_0]{+} w_1 D_j \xRightarrow[G_0]{+} w'_1 w'_2 = w$$

is valid.

This completes the proof of the claim.

To conclude the proof of the lemma, let us check that G_0 genetates the nonunary part of $L(G)$. For the direct implication, assume that $u \in L(G_0)$. Its derivation should be

$$S' \xRightarrow[G_0]{} B_i C_j \xRightarrow[G_0]{+} u_1 C_j \xRightarrow[G_0]{+} u_1 u_2 = u$$

for some $S' \to B_i C_j \in P'$, $u_1, u_2 \in \Sigma^+$, $1 \le i, j \le m$, with $i \ne j$. Ignoring the subscripts i, j in this derivation brings us with $S \xRightarrow[G]{+} u$. Moreover, the claim above implies that u_1 and u_2 contain a_i and a_j, respectively. Thus, u is a nonunary word in $L(G)$.

Conversely, consider a nonunary word $w \in L(G)$. Being nonunary, $|w| \geq 2$, and this means that its derivation by G must begin with a production $S \to BC$. Since $B, C \neq S$, they cannot produce ε, and hence, we have

$$S \underset{G}{\Longrightarrow} BC \overset{+}{\underset{G}{\Longrightarrow}} w_1 C \overset{+}{\underset{G}{\Longrightarrow}} w_1 w_2 = w$$

for some nonempty words $w_1, w_2 \in \Sigma^+$. Again, being w nonunary, we can find a letter a_i in w_1 and a letter a_j in w_2 such that $i \neq j$. Now, the claim above implies $B_i \overset{+}{\underset{G_0}{\Longrightarrow}} w_1$ and $C_j \overset{+}{\underset{G_0}{\Longrightarrow}} w_2$. Since $S' \to B_i C_j \in P'$, the derivation

$$S' \underset{G_0}{\Longrightarrow} B_i C_j \overset{+}{\underset{G_0}{\Longrightarrow}} w_1 C_j \overset{+}{\underset{G_0}{\Longrightarrow}} w_1 w_2 = w$$

is a valid one by G_0.

Note that, being thus designed, G_0 contains $mh - m + 1$ variables. □

Let us see with an example how the lemma proceeds in the case of cfgs. Here we report an example to obtain the nonunary part of a language generated by a Cnfg.

Example 3.4.2. *Let us consider a* Cnfg $G = (V, \Sigma, P, S)$*, where*

$$V = \{S, A, B, C, D, E\}$$

and P consists of the following production rules:

1. $\{S \to AC \mid BD \mid a \mid b \mid \varepsilon\}$;
2. $\{A \to a\}$;
3. $\{B \to b\}$;
4. $\{C \to EA \mid a\}$;
5. $\{D \to EB \mid b\}$;
6. $\{E \to AC \mid BD \mid a \mid b\}$;

generating the language of all palindrome words in $\Sigma = \{a, b\}$.

Now, we give the Cnfg G_0 *as a quadruple* (V', Σ, P', S')*, where*

$$V' = \{S'\} \cup \{A_1, A_2, B_1, B_2, C_1, C_2, D_1, D_2, E_1, E_2\}$$

and P' consists of the following production rules:

1. $\{S' \to A_1 C_2 \mid A_2 C_1 \mid B_1 D_2\}$;
2. $\{A_1 \to a\}$;

3. $\{B_2 \to b\}$;

4. $\{C_2 \to E_2A_1 \mid E_2A_2 \mid E_1A_2\}$;

5. $\{C_1 \to E_2A_1 \mid E_1A_2 \mid E_1A_1 \mid a\}$;

6. $\{D_1 \to E_1B_1 \mid E_1B_2 \mid E_2B_1\}$;

7. $\{D_2 \to E_2B_1 \mid E_2B_2 \mid E_1B_2 \mid b\}$;

8. $\{E_1 \to A_1C_1 \mid A_1C_2 \mid A_2C_1 \mid B_1D_1 \mid B_1D_2 \mid B_2D_1 \mid a\}$;

9. $\{E_2 \to A_2C_2 \mid A_2C_1 \mid A_1C_2 \mid B_2D_2 \mid B_2D_1 \mid B_1D_2 \mid b\}$;

□

We conclude this section by shortly discussing some constructions related to the union of languages defined by 1dfas and by 2dfas.

3.4.2 Composition results

First, we remind the reader that k 1dfas $A_1, A_2, \ldots, A_k$ with $n_1, n_2, \ldots, n_k$ states, respectively, can be simulated “in parallel” by a 1dfa, in order to recognize the union $L(A_1) \cup L(A_2) \cup \cdots \cup L(A_k)$. In particular, the state set of A is the Cartesian product of the state sets of the given automata. For this reason, the automaton A obtained according to this standard construction is usually called *product automaton*. Its number of states is $n_1 \cdot n_2 \cdots n_k$.

If $A_1, A_2, \ldots, A_k$ are 2dfas and we want to obtain a 2dfa A accepting the union $L(A_1) \cup L(A_2) \cup \cdots \cup L(A_k)$, the state cost reduces to the sum $n_1 + n_2 + \cdots + n_k$, *under the hypothesis that the automata are halting*, namely, they do not present any infinite computation. We observe that the drastic decrease in state complexity is due to the capability of moving the reading head in both directions. In particular, on input w, the automaton A simulates in sequence, for $i = 1, \ldots, k$, the automata A_i, halting and accepting in the case one $\hat{\imath}$ is found such that $A_{\hat{\imath}}$ accepts w.

Suppose that, for $i = 1, \ldots, k$, the state set of A_i is Q_i with final state $q_{i,f}$ and $Q_i \cap Q_j \neq \emptyset$ for $i \neq j$. Then, A can be defined as follows.

- The set of states is $Q = Q_1 \cup Q_2 \cup \cdots \cup Q_k$.
- The initial state is the initial state of A_1.
- The final state is the final state $q_{k,f}$ of A_k.
- For $i = 1, \ldots, k-1$, A contains all the transitions of A_i with the exception of those leading to the final state $q_{i,f}$ of A_i. Those transitions lead directly to $q_{k,f}$, to halt and accept.

In this way, the state $q_{i,f}$ of A_i becomes unreachable. (We remind the reader that this state is also halting.) In the automaton A, the state $q_{i,f}$ is “recycled” in a different way: it is used to prepare the simulation of A_{i+1} after a rejecting simulation of A_i. To this aim, each undefined transition of A_i leads in A to the state $q_{i,f}$, where the automaton A loops, moving the head leftward, to reach the left endmarker. There, A moves the head one position to the right, on the first symbol of the input word, and enters the initial state of A_{i+1}, hence starting to simulate it.

- All the transitions of A_k are copied in A without any change. Hence, if the input was rejected in all the simulations of $A_1, A_2, \ldots, A_{k-1}$, it is accepted by A if and only if it is accepted by A_k.

We observe that each 1dfa can be converted into a 2dfa in the form we are considering (cf. Definition 2.5.2), just adding the accepting state, which is entered on the right endmarker when the given 1dfa accepts the input. So the above construction works (with the addition of at most k extra states) even when some of the A_i's are one-way.

Finally, we point out that, as proven in [GMP07], with a linear increase in the number of the states, each 2dfas can be made halting. In particular, each n-state 2dfa can be simulated by a halting 2dfa with $4n$ states.

So the above outlined construction can be extended to the case of nonhalting 2dfas by obtaining a 2dfa with no more than $4 \cdot (n_1 + n_2 + \cdots + n_k)$ states.

3.5 State complexity of operations on regular languages

In this section we recall some results about state complexity of some operations between regular languages that we will use in Chapter 6. In particular, we recall the state complexity of operations union, intersection, complement, concatenation, star [Yu00], shuffle [CSY02], reversal [YZS94] and projection [JM12].

We assume that L_1 is an n_1-state 1dfa language and L_2 is an n_2-state 1dfa language, and $n_1, n_2 > 1$. Table 3.1 shows the state complexity of the operations over unary and nonunary alphabet.

Operation	Unary alphabet	Nonunary alphabet
$L_1 \cup L_2$	$n_1 n_2$	$n_1 n_2$
$L_1 \cap L_2$	$n_1 n_2$	$n_1 n_2$
L_1^c	n_1	n_1
$L_1 L_2$	$n_1 n_2$	$(2n_1 - 1)2^{n_2 - 1}$
L_1^*	$(n_1 - 1)^2 + 1$	$2^{n_1 - 1} + 2^{n_1 - 2}$
$L_1 \sqcup\!\sqcup L_2$	$n_1 n_2$	$2^{n_1 n_2} - 1$
L_1^R	n_1	2^{n_1}
$P_{\Sigma_0}(L_1)$	n_1	$3 \cdot 2^{n_1 - 2} - 1$

Table 3.1: Operational state complexity on regular languages accepted by 1dfas.

Chapter 4

Semilinear sets and Parikh's Theorem

Die Walküre

Walkürenritt
Richard Wagner

In this chapter we recall some notions concerning semilinear set and Parikh's theorem. In particular, we obtain a normal form for the Parikh image of languages accepted by 1nfas (Lemma 4.0.2) which is a fundamental tool used in the next chapters.

Let us start with some preliminary notions.

Given k vectors $\boldsymbol{v_1}, \ldots, \boldsymbol{v_k} \in \mathbb{Z}^m$, we say that they are *linearly independent* if for all $n_1, \ldots, n_k \in \mathbb{Z}$, $\boldsymbol{v_1} n_1 + \ldots + \boldsymbol{v_k} n_k = \mathbf{0}$ implies $n_1 = \ldots = n_k = 0$. It is well known that, in this case, k cannot exceed m. The following result will be required later.

Lemma 4.0.1. *Given k linearly independent vectors $\boldsymbol{v_1}, \ldots, \boldsymbol{v_k} \in \mathbb{Z}^m$ there are k pairwise different integers $t_1, \ldots, t_k \in \{1, \ldots, m\}$ such that $\boldsymbol{v_j}[t_j] \neq 0$, for $j = 1, \ldots, k$.*

Proof. Let W be the $m \times k$ matrix which has $\boldsymbol{v_1}, \ldots, \boldsymbol{v_k}$ as columns. Since the given vectors are linearly independent, $k \leq m$. Furthermore, by suitably deleting $m - k$ rows from W, we can obtain a $k \times k$ matrix V whose determinant $d(V)$ is nonnull.

If $k = 1$ then the result is trivial. Otherwise, we can compute $d(V)$ along the last column as

$$d(V) = \sum_{i=1}^{k} (-1)^{i+k} \boldsymbol{v_k}[i] d_{i,k}$$

where $d_{i,k}$ is the determinant of the matrix $V_{i,k}$ obtained by removing from V the row i and the column k. Since $d(V) \neq 0$, there is at least one index

i such that $\boldsymbol{v_k}[i] \neq 0$ and $d_{i,k} \neq 0$. Hence, as t_k we take such i. Using an induction on the matrix $V_{t_k,k}$, we can finally obtain the sequence $t_1, \ldots, t_k$ satisfying the statement of the theorem. □

A vector $\boldsymbol{v} \in \mathbb{Z}^m$ is *unary* if it contains at most one nonzero component, i.e., $\boldsymbol{v}[i], \boldsymbol{v}[j] \neq 0$ for some $1 \leq i, j \leq m$ implies $i = j$; otherwise, it is *nonunary*. By definition, the null vector is unary.

We define $\preceq$ for the componentwise partial order on $\mathbb{N}^m$, i.e., $\boldsymbol{u} \preceq \boldsymbol{v}$ if and only if $\boldsymbol{u}[k] \leq \boldsymbol{v}[k]$ for all $1 \leq k \leq m$. For a vector $\boldsymbol{v} \in \mathbb{N}^m$, let

$$\mathrm{Pred}(\boldsymbol{v}) = \{\boldsymbol{u} \mid \boldsymbol{u} \preceq \boldsymbol{v}\}\,.$$

For $\boldsymbol{u}, \boldsymbol{v} \in \mathbb{N}^m$, $\boldsymbol{v} - \boldsymbol{u}$ is defined to be a vector $\boldsymbol{w}$ with $\boldsymbol{w}[k] = \boldsymbol{v}[k] - \boldsymbol{u}[k]$ for all $1 \leq k \leq m$. Note that $\boldsymbol{v} - \boldsymbol{u}$ is a vector in $\mathbb{N}^m$ if and only if $\boldsymbol{u} \preceq \boldsymbol{v}$.

A *linear set* in $\mathbb{N}^m$ is a set of the form

$$\{\boldsymbol{v}_0 + n_1\boldsymbol{v}_1 + n_2\boldsymbol{v}_2 + \cdots + n_k\boldsymbol{v}_k \mid n_1, n_2, \ldots, n_k \in \mathbb{N}\}\ , \tag{4.1}$$

where $k \geq 0$ and $\boldsymbol{v}_0, \boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_k \in \mathbb{N}^m$. The vector $\boldsymbol{v}_0$ is called *offset* (a.k.a. *constant*), while the vectors $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_k$ are called *generators* (a.k.a. *periods*). A *semilinear set* in $\mathbb{N}^m$ is a finite union of linear sets in $\mathbb{N}^m$.

From Theorem 6.1, Corollary 1 of Theorem 6.2, and Lemma 6.3 of [GS64] there follows:

Theorem 4.0.1. *The family of semilinear sets of* $\mathbb{N}^m$ *is closed under union, intersection, and complementation. The projection of a semilinear set is semilinear.*

The *Parikh map* $\psi : \Sigma^* \to \mathbb{N}^m$ associates with a word $w \in \Sigma^*$ the vector

$$\psi(w) = (|w|_{a_1}, |w|_{a_2}, \ldots, |w|_{a_m})\ ,$$

which counts the occurrences of each letter of Σ in w. The vector $\psi(w)$ is also called *Parikh image* of w. Notice that a word $w \in \Sigma^*$ is *unary* if and only if its Parikh image $\psi(w)$ is a unary vector. One can naturally generalize this map for a language $L \subseteq \Sigma^*$ as

$$\psi(L) = \{\psi(w) \mid w \in L\}\,.$$

The set $\psi(L)$ is called the *Parikh image* of L. Two languages $L, L' \subseteq \Sigma^*$ are said to be *Parikh equivalent* if $\psi(L) = \psi(L')$.

Parikh equivalence can be defined not only between languages but among languages, grammars, and finite automata by referring, in the last two cases, to the defined languages. For example, given a language L, a cfg G, and a finite automaton A, we say that:

- G is Parikh equivalent to L if $\psi(L(G)) = \psi(L)$,

- A is Parikh equivalent to L if $\psi(L(A)) = \psi(L)$,
- G is Parikh equivalent to A if $\psi(L(G)) = \psi(L(A))$.

Parikh's Theorem, proven in 1966 [Par66], states that *the Parikh image of any context-free language is a semilinear set.* We observe that the converse of the theorem is not true: the Parikh image of the language $\{a^n b^n c^n \mid n \geq 0\}$ is a semilinear set, but the language is not context free. Furthermore, we can prove that a language is not context free using the contrapositive of the Parikh's Theorem: if $\psi(L)$ is not a semilinear set then L is not context free. For instance, $\psi(L) = \left\{(n^2) \mid n \geq 1\right\}$ where $L = \left\{a^{n^2} \mid n \geq 1\right\}$ cannot be expressed as a semilinear set.

Since the class of regular languages is closed under union and each linear set as in (4.1) is the Parikh image of the regular language

$$\{w_0\} \cdot \{w_1, w_2, \ldots, w_k\}^*,$$

where, for $i = 0, \ldots, k$, $w_i = a_1^{\boldsymbol{v}_i[1]} a_2^{\boldsymbol{v}_i[2]} \cdots a_m^{\boldsymbol{v}_i[m]}$, Parikh's Theorem is frequently formulated by giving the following immediate consequence:

Theorem 4.0.2 ([Par66])**.** *Every context-free language is Parikh equivalent to a regular language.*

It is immediate to observe that in the case of unary languages, i.e., languages defined over a one letter alphabet, two languages are Parikh equivalent if and only if they are equal. Hence, as a consequence of Theorem 4.0.2, each unary context-free language is regular. This result was firstly proved, without using Parikh's Theorem, by Ginsburg and Rice [GR62]. The equivalence between unary context-free and regular languages has been studied from the descriptional complexity point of view in [PSW02], where the following result was proved:

Theorem 4.0.3 ([PSW02, Thms. 4, 6, 5, 7])**.** *For any* Cnfg *with h variables that generates a unary language, there exist an equivalent* 1nfa *with at most $2^{2h-1}+1$ states and an equivalent* 1dfa *with less than 2^{h^2} states. Furthermore, these costs are tight.*

In the thesis we will also make use of the transformation of unary 1nfas into 1dfas, whose cost was obtained in 1986 by Chrobak [Chr86]:

Theorem 4.0.4 ([Chr86])**.** *The state cost of the conversion of n-state unary* 1nfa*s into equivalent* 1dfa*s is $e^{\Theta(\sqrt{n \cdot \ln n})}$.*

Actually, Geffert observed that a more precise value is $e^{(1+o(1))\sqrt{n \cdot \ln n}}$ [Gef07].

From now on, let us fix an alphabet $\Sigma = \{a_1, a_2, \ldots, a_m\}$. A fundamental tool which will be used is the following normal form for the Parikh image of 1nfas, which is based on a result obtained by Kopczyński and To in 2010 [KT10].

Lemma 4.0.2. *There exists a polynomial p such that for each n-state* 1nfa *A over Σ, the Parikh image of the language accepted by A can be written as*

$$\psi(L(A)) = Y \cup \bigcup_{i \in I} Z_i\,, \tag{4.2}$$

where:

- *$Y \subseteq \mathbb{N}^m$ is a finite set of vectors whose components are bounded by $p(n)$;*
- *I is a set of at most $p(n)$ indices;*
- *for each $i \in I$, $Z_i \subseteq \mathbb{N}^m$ is a linear set of the form:*

  $$Z_i = \{\boldsymbol{v}_{i,0} + n_1\boldsymbol{v}_{i,1} + n_2\boldsymbol{v}_{i,2} + \cdots + n_{k_i}\boldsymbol{v}_{i,k_i} \mid n_1, n_2, \ldots, n_{k_i} \in \mathbb{N}\}\,, \tag{4.3}$$

  with:

  – $0 \le k_i \le m$,

  – *the components of the offset $\boldsymbol{v}_{i,0}$ are bounded by $p(n)$,*

  – *the generators $\boldsymbol{v}_{i,1}, \boldsymbol{v}_{i,2}, \ldots, \boldsymbol{v}_{i,k_i}$ are linearly independent vectors from $\{0, 1, \ldots, n\}^m$.*

Furthermore, if all the words in $L(A)$ are nonunary then, for each $i \in I$ we can choose a nonunary vector $\boldsymbol{x}_i \in \mathrm{Pred}(\boldsymbol{v}_{i,0})$ such that all those chosen vectors are pairwise distinct.

Proof. In [KT10, Thms. 7 and 8] it was proved that $\psi(L(A))$ can be written as claimed in the first part of the statement of the lemma, with $Y = \emptyset$, I of size polynomial in n, and the components of each offset $\boldsymbol{v}_{i,0}$ bounded by $O(n^{2m}m^{m/2})$.

For the sake of completeness, we derive a rough upper bound for the cardinality of the set I by counting the number of possible combinations of offsets and generators satisfying the given limitations. Since the components of the offsets are bounded by $O(n^{2m}m^{m/2})$, the number of possible different offsets is $O(n^{2m^2}m^{m^2/2})$. We can observe that due to the linear independence of the generators we have $k \le m$. Furthermore, there are $(n+1)^m$ vectors in $\{0, 1, \ldots, n\}^m$. Hence, n^{m^2} is an upper bound for the number of possible sets of k generators, with $k = 1, \ldots, m$. This allows us to give $O(n^{3m^2}m^{m^2/2})$ as an upper bound for the cardinality of I. We point out that in [To10b, Thm. 4.1] slightly different bounds have been given.

Now we prove the second part of the statement. Hence, let us suppose that all the words in $L(A)$ are nonunary. Notice that this implies that also all the offsets $\boldsymbol{v}_{i,0}$ are nonunary.

If for each $i \in I$ we can choose $\boldsymbol{x}_i \in \mathrm{Pred}(\boldsymbol{v}_{i,0})$ such that all $\boldsymbol{x}_i$'s are pairwise different, then the proof is completed.

Otherwise, we proceed as follows. For a vector $\boldsymbol{v}$, let us denote by $\|\boldsymbol{v}\|$ its *infinite norm*, i.e., the value of its maximum component. Let us suppose $I \subseteq \mathbb{N}$ and denote as N_I the maximum element of I.

By proceeding in increasing order, for $i \in I$ we choose a nonunary vector $\boldsymbol{x}_i \in \mathrm{Pred}(\boldsymbol{v}_{i,0})$ such that $\|\boldsymbol{x}_i\| \leq i$ and $\boldsymbol{x}_i$ is different from all already chosen $\boldsymbol{x}_j$, i.e., $\boldsymbol{x}_i \neq \boldsymbol{x}_j$ for all $j \in I$ with $j < i$. The extra condition $\|\boldsymbol{x}_i\| \leq i$ will turn out to be useful later.

When for an $i \in I$ it is not possible to find such $\boldsymbol{x}_i$, we replace Z_i by some suitable sets. Essentially, those sets are obtained by enlarging the offsets using sufficiently long "unrollings" of the generators. In particular, for $j = 1, \ldots, k_i$, we consider the set

$$Z_{N_I+j} = \{(\boldsymbol{v}_{i,0} + h_j \boldsymbol{v}_{i,j}) + n_1 \boldsymbol{v}_{i,1} + \cdots + n_{k_i} \boldsymbol{v}_{i,k_i} \mid n_1, \ldots, n_{k_i} \in \mathbb{N}\}\,, \quad (4.4)$$

where h_j is an integer satisfying the inequalities

$$N_I + j \leq \|\boldsymbol{v}_{i,0} + h_j \boldsymbol{v}_{i,j}\| < N_I + j + n \quad (4.5)$$

Due to the fact that $\boldsymbol{v}_{i,j} \in \{0, \ldots, n\}^m$, we can always find such h_j. Furthermore, we consider the following finite set

$$Y_i = \{\boldsymbol{v}_{i,0} + n_1 \boldsymbol{v}_{i,1} + \cdots + n_{k_i} \boldsymbol{v}_{i,k_i} \mid 0 \leq n_1 < h_1, \ldots, 0 \leq n_{k_i} < h_{k_i}\}\,. \quad (4.6)$$

It can be easily verified that

$$Z_i = Y_i \cup \bigcup_{j=1}^{k_i} Z_{N_I+j}\,.$$

Now we replace the set of indices I by the set

$$\widehat{I} = I - \{i\} \cup \{N_I + 1, \ldots, N_I + k_i\}\,,$$

and the set Y by $\widehat{Y} = Y \cup Y_i$. We continue the same process by considering the next index i.

We notice that, since we are choosing each vector $\boldsymbol{x}_i \in \mathrm{Pred}(\boldsymbol{v}_{i,0})$ in such a way that $\|\boldsymbol{x}_i\| \leq i$, when we will have to choose the vector $\boldsymbol{x}_{N_I+j}$ for a set Z_{N_I+j} introduced at this stage, by the condition (4.5) we will have at least one possibility (a vector with one component equal to $N_I + j$ and another component equal to 1; we remind the reader that, since the given automaton accepts only nonunary words, all offsets are nonunary). This implies that after examining all sets Z_i corresponding to the original set I, we do not need to further modify the sets introduced during this process. Hence, this procedure ends in a finite number of steps.

Furthermore, for each Z_i in the initial representation, we introduced at most m sets. Since the cardinality of the set of indices I, before the transformation was $O(n^{3m^2} m^{m^2/2})$, the cardinality $\widetilde{N}$ after the transformation is $O(n^{3m^2} m^{m^2/2+1})$. Hence, the cardinality $\widetilde{N}$ of the set of indices resulting at the end of this process is still polynomial.

By (4.5) the components of the offsets which have been added in this process cannot exceed $\widetilde{N} + n$. Hence, it turns out that $m \cdot (\widetilde{N} + n)$ is an upper bound to the components of vectors in Y_i. This permits to conclude that $p(n) = m \cdot (\widetilde{N} + n)$ is an upper bound for all these amounts. Hence $p(n) = O(n^{3m^2} m^{m^2/2+2})$. □

Chapter 5

Conversions under Parikh equivalence

Veni Creator Spiritus

Symphony No. 8 in E-flat Major
Symphony of a Thousand
Gustav Mahler

In this chapter we investigate conversions from one-way nondeterministic automata and context-free grammars to one-way and two-way deterministic automata according to Parikh equivalence.

5.1 Conversions into Parikh equivalent 1dfas

Here we investigate conversions from one-way nondeterministic automata and context-free grammars to one-way deterministic automata according to Parikh equivalence. Conversions from context-free grammars are treated in Section 5.1.2, while we start presenting conversions from one-way nondeterministic automata.

5.1.1 From 1nfas to Parikh equivalent 1dfas

In this section we present our first main contribution. From each n-state 1nfa A, we derive a Parikh equivalent 1dfa A' with $e^{O(\sqrt{n \cdot \ln n})}$ states. Furthermore, we prove that this cost is tight.

Actually, as a preliminary step, we obtain a result which is interesting *per se*: if each word accepted by the given 1nfa A contains at least two different symbols, i.e., it is nonunary, then the Parikh equivalent 1dfa A' can be obtained with polynomially many states. Hence, the superpolynomial blowup is due to the unary part of the accepted language. This result looks quite surprising.

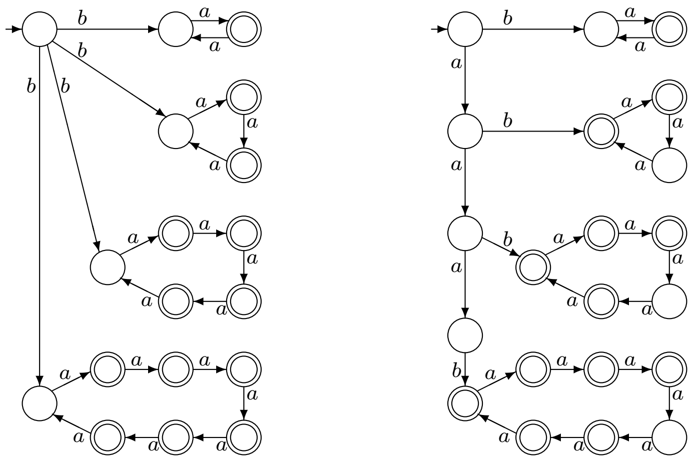

Figure 5.1: *(Left)* The 1nfa A of Example 5.1.1. *(Right)* The Parikh equivalent 1dfa A'.

Before starting the technical presentation, we show an example with the aim to give, in a very simple case, a taste of our constructions.

Example 5.1.1. *Let us consider the following language*

$$L = \{ba^n \mid n \bmod 210 \neq 0\} .$$

Clearly, L does not contain any unary word. Furthermore, it can be verified that L is accepted by the 18*-state* 1nfa *A in Figure 5.1*(Left)*. In particular, in the initial state, reading the letter b,* in a nondeterministic way *A chooses to verify the membership of the input to one of the following languages:*

- $L_1 = \{ba^n \mid n \bmod 2 \neq 0\}$,
- $L_2 = \{ba^n \mid n \bmod 3 \neq 0\}$,
- $L_3 = \{ba^n \mid n \bmod 5 \neq 0\}$,
- $L_4 = \{ba^n \mid n \bmod 7 \neq 0\}$.

Of course, $L = L_1 \cup L_2 \cup L_3 \cup L_4$. The automaton A can be transformed into an equivalent 1dfa*, by identifying the transitions leaving the initial state and by merging the 4 loops into a unique loop of length* $2 \cdot 3 \cdot 5 \cdot 7 = 210$*. Using standard distinguishability arguments, it can be shown that it is not possible to do better. As a matter of fact, the smallest complete* 1dfa *accepting L requires 212 states.*

However, we can build a complete 1dfa *A' with only* 22 *states, accepting a language L' Parikh equivalent to L. To do that, for $i = 1, \dots, 4$, we replace each language L_i with a Parikh equivalent language L'_i in such a way that all the words in L'_i begin with the prefix $a^{i-1}b$, and then we define L' as the union of the resulting languages, namely:*

- $L'_1 = \{ba^n \mid n \bmod 2 \neq 0\} = L_1$,
- $L'_2 = \{aba^{n-1} \mid n \bmod 3 \neq 0\}$,
- $L'_3 = \{a^2ba^{n-2} \mid n \bmod 5 \neq 0\}$,
- $L'_4 = \{a^3ba^{n-3} \mid n \bmod 7 \neq 0\}$,
- $L' = L'_1 \cup L'_2 \cup L'_3 \cup L'_4$.

In this way, given an input word w, after reading the first 4 input symbols, in a deterministic way *A' can decide for which language L'_i, $1 \leq i \leq 4$, the membership of w should be tested, in order to decide whether or not $w \in L'$.*

The automaton A' is depicted in Figure 5.1(Right)*. The vertical path starting from the initial state is used to select, depending on the position of the letter b, one of the loops, i.e., to select which language L'_i must be used to decide the membership of the input to L'. (Of course, when the symbol b does not appear in the prefix of length* 4*, the automaton rejects by entering a dead state, which is not depicted.)*

The loops of A' are obtained by suitably "unrolling" the loops of the original 1nfa *A. The unrolled parts of the loops are moved before b-transitions and merged together in the vertical path which starts from the initial state.* □

A fundamental tool which will be used in this section is the normal form for the Parikh image of 1nfas (presented in Lemma 4.0.2).

Now we are able to consider the case of automata accepting only words that are nonunary.

Theorem 5.1.1. *For each n-state* 1nfa *over an m-letter alphabet Σ, accepting a language none of whose words are unary, there exists a Parikh equivalent* 1dfa *with at most $O(n^{3m^3+6m^2}m^{m^3/2+m^2+2m+5})$ states.*

Proof. Let A be the given n-state 1nfa. According to Lemma 4.0.2, we express the Parikh image of $L(A)$ as:

$$\psi(L(A)) = Y \cup \bigcup_{i \in I} Z_i \,,$$

and, starting from this representation, we will build a 1dfa A_{non} that is Parikh equivalent to A. To this end, we could apply the following procedure:

1. For each $i \in I$, build a 1dfa A_i such that $\psi(L(A_i)) = Z_i$.
2. From the automata A_i's so obtained, derive a 1dfa A' such that $\psi(L(A')) = \bigcup_{i \in I} Z_i$.
3. Define a 1dfa A'' such that $\psi(L(A'')) = Y$.

4. From A' and A'', using the standard construction for the union, build a 1dfa A_{non} such that $L(A_{\mathrm{non}}) = L(A') \cup L(A'')$ and, hence, $\psi(L(A_{\mathrm{non}})) = Y \cup \bigcup_{i \in I} Z_i$, i.e., A_{non} is Parikh equivalent to A.

Actually, we will use a variation of this procedure. In particular, steps 1 and 2, to obtain A', are modified as we now explain.

Let us start by considering $i \in I$. First, we handle the generators of Z_i. To this aim, let us consider the function $g : \mathbb{N}^m \to \Sigma^*$ defined by

$$g(\boldsymbol{v}) = a_1^{i_1} a_2^{i_2} \cdots a_m^{i_m} ,$$

for each vector $\boldsymbol{v} = (i_1, \ldots, i_m) \in \mathbb{N}^m$.

Using this function, we map the generators $\boldsymbol{v}_{i,1}, \ldots, \boldsymbol{v}_{i,k_i}$ into the words

$$s_{i,1} = g(\boldsymbol{v}_{i,1}),\ s_{i,2} = g(\boldsymbol{v}_{i,2}),\ \ldots,\ s_{i,k_i} = g(\boldsymbol{v}_{i,k_i}) .$$

It is easy to define an automaton accepting the language $\{s_{i,1}, s_{i,2}, \ldots, s_{i,k_i}\}^*$, which consists of a start state q with k_i loops labeled with $s_{i,1}, s_{i,2}, \ldots, s_{i,k_i}$, respectively. The state q is the only accepting state. However, this automaton is nondeterministic.

To avoid this problem, we modify the language by replacing each $s_{i,j}$, for $j = 1, \ldots, k_i$, with a Parikh equivalent word $w_{i,j}$ in such a way that for all pairwise different j, j' the corresponding words $w_{i,j}$ and $w_{i,j'}$ begin with different letters. This is possible due to the fact, being $\boldsymbol{v}_{i,1}, \ldots, \boldsymbol{v}_{i,k_i}$ linearly independent, according to Lemma 4.0.1 we can find k_i different letters $a_{t_1}, a_{t_2}, \ldots, a_{t_{k_i}} \in \Sigma$ such that $\boldsymbol{v}_{i,j}[t_j] > 0$ for $j = 1, \ldots, k_i$. We "rotate" each $s_{i,j}$ by a cyclic shift so that the resulting word, $w_{i,j}$, begins with an occurrence of the letter a_{t_j}. Then $w_{i,j}$ is Parikh equivalent to $s_{i,j}$. For example, if $s_{i,j} = a_1^3 a_2^4 a_3$ and $t_j = 2$, then $w_{i,j}$ should be chosen as $a_2^4 a_3 a_1^3$.

The construction of a 1dfa B_i with one unique accepting state q that accepts $\{w_{i,1}, w_{i,2}, \ldots, w_{i,k_i}\}^*$ must be now clear: q with k_i loops labeled with these respective k_i words (see Figure 5.2). Furthermore, due to the limitations deriving from Lemma 4.0.2, the length of these loops is at most mn so that this 1dfa contains at most $1 + m(mn - 1)$ states.

Now, we can modify this 1dfa in order to build an automaton A_i recognizing a language whose Parikh image is Z_i. To this aim, it is enough to add a path which from an initial state, reading a word $w_{i,0}$ with Parikh image $\boldsymbol{v}_{i,0}$, reaches the state q and then the part accepting $\{w_{i,1}, w_{i,2}, \ldots, w_{i,k_i}\}^*$ that we already described. Due to the limitations on $\boldsymbol{v}_{i,0}$ from Lemma 4.0.2, this can be done by adding a polynomial number of states. In particular, we could take $w_{i,0} = g(\boldsymbol{v}_{i,0})$, thus completing step 1. However, when we have such 1dfas for all $i \in I$, by applying the standard construction for the union to them, as in step 2, being I polynomial in n, the resulting 1dfa could have exponentially many states in n, namely it could be too large for our purposes.

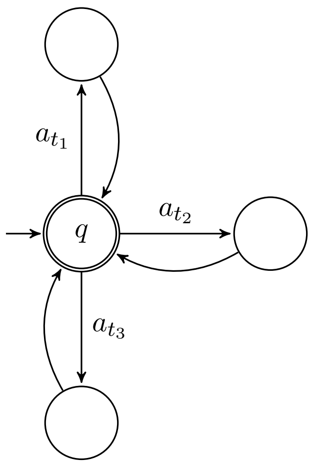

Figure 5.2: A construction of 1dfa B_i that accepts $\{w_{i,1}, w_{i,2}, \ldots, w_{i,k_i}\}^*$ for $k_i = 3$.

To avoid this problem, the automaton A' which should be derived from steps 1-2, is obtained by using a different strategy. We introduce the function $f : \mathbb{N}^m \to \Sigma^*$ defined as: for $\boldsymbol{v} \in \mathbb{N}^m$, $f(\boldsymbol{v}) = \hookleftarrow(g(\boldsymbol{v}))$, where $\hookleftarrow$ denotes the 1-step left circular shift. For example, $f(4,1,2,0,\ldots,0) = a_1^3 a_2 a_3^2 a_1$. It can be verified that the 1-step left circular shift endows f with the prefix property *over the nonunary vectors*, that is, for any $\boldsymbol{u}, \boldsymbol{v} \in \mathbb{N}^m$ that are nonunary, if $f(\boldsymbol{u})$ is a prefix of $f(\boldsymbol{v})$, then $\boldsymbol{u} = \boldsymbol{v}$. Let

$$w_{i,0} = f(\boldsymbol{x}_i) g(\boldsymbol{v}_{i,0} - \boldsymbol{x}_i)\,,$$

where $\boldsymbol{x}_i \in \mathrm{Pred}(\boldsymbol{v}_{i,0})$ is given by Lemma 4.0.2. Clearly, $\psi(w_{i,0}) = \boldsymbol{v}_{i,0}$. We now consider the finite language

$$W = \{w_{i,0} \mid i \in I\}\,.$$

Because the $\boldsymbol{x}_i$'s are nonunary and f has the prefix property over nonunary words, the language W is prefix-free. We build a (partial) 1dfa that accepts W, which is denoted by $A_W = (Q_W, \Sigma, q_\varepsilon, \delta_W, F_W)$, where:

- $Q_W = \{q_u \mid u \in \mathrm{Pref}(W)\}$,
- the state q_ε corresponding to the empty word is the initial state,
- $F_W = \{q_u \mid u \in W\}$,
- δ_W is defined as: for $u \in \mathrm{Pref}(W)$ and $a \in \Sigma$, if $ua \in \mathrm{Pref}(W)$, then $\delta(q_u, a) = q_{ua}$, while $\delta(q_u, a)$ is undefined otherwise.

See Figure 5.3. In the construction of the final 1dfa A_{non}, if $w_{i,0} = acb$, then the initial state q of B_i is merged with the state q_{acb} of A_W. Clearly, this accepts W. Since the longest word(s) in W is of length $m \cdot p(n)$, this 1dfa contains at most $1 + |I| \cdot m \cdot p(n) = O(mp^2(n))$ states.

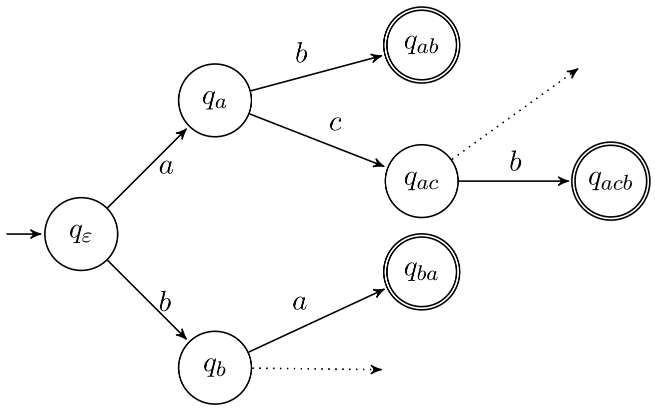

Figure 5.3: A construction of 1dfa A_W.

It goes without saying that each accepting state of this 1dfa is only for one word in W, namely, two distinct words in W are accepted by A_W at distinct two states. Now, based on A_W and the 1dfas B_i with $i \in I$, we can build a finite automaton that accepts the language $\bigcup_{i\in I} w_{i,0}L(B_i)$ *without introducing any new state.* This is simply done by merging $q_{w_{i,0}}$ with the start state of B_i. Given an input u, the resulting automaton A' simulates the 1dfa A_W, looking for a prefix w of u such that $w \in W$. When such a prefix is found, A' starts simulating B_i on the remaining suffix z, where i is the index such that $w = w_i$. Since W is prefix-free, we need only to consider one decomposition of the input as $u = wz$. This implies that A' is deterministic. Finally, we observe that A' contains at most $O(mp^2(n)) + |I|(1 + m(mn - 1)) = O(mp(n)(p(n) + mn))$ states, i.e., a number which is polynomial in n.

We now sketch the construction of a 1dfa A'' accepting a language L_Y whose Parikh image is Y (step 3). We just take $L_Y = \{g(\boldsymbol{v}) \mid \boldsymbol{v} \in Y\}$. Let M be the maximum of the components of vectors in Y. With each $\boldsymbol{v} \in \{0, \ldots, M\}^m$, we associate a state $q_{\boldsymbol{v}}$ which is reachable from the initial state by reading the word $g(\boldsymbol{v})$. Final states are those corresponding to vectors in Y. The automaton A'' so obtained has $(M + 1)^m = (p(n) + 1)^m$ states, a number polynomial in n.

Finally, by applying the standard construction for the union (step 4), from automata A' and A'' we obtain the 1dfa A_{non} Parikh equivalent to the given 1nfa A, with number of states polynomial in n.

In fact, assuming $p(n) \geq mn$, the number of states of A' is $O(mp^2(n))$.

Hence, the number of states of A_{non} is $O(mp^{m+2}(n))$. Using $p(n) = O(n^{3m^2}m^{m^2/2+2})$ (from Lemma 4.0.2), we conclude that the number of state of A_{non} is $O(mn^{3m^2(m+2)}m^{(m^2/2+2)(m+2)}) = O(n^{3m^3+6m^2}m^{m^3/2+m^2+2m+5})$.

Hence, it is polynomial in the number of states of the original 1nfa A, but exponential in the alphabet size m. □

It should be interesting to see whether or not the cost for the conversion in the case n-state 1nfas accepting only nonunary words could be further reduced. To this respect, we point out that n is a lower bound. In fact, a smaller cost would imply that any given 1nfa (or 1dfa) B_0, could be converted into a smaller Parikh equivalent 1dfa B_1 which, in turn, could be further converted in a smaller Parikh equivalent 1dfa B_2 an so on. In this way, from B_0 we could build an arbitrary long sequence of automata $B_0, B_1, B_2, \ldots$, all of them Parikh equivalent to B_0, and such that for each $i > 0$, B_i would be smaller than B_{i-1}. This clearly does not make sense. With a similar argument, we can also conclude that even the costs of the conversion of n-state 1nfas into a Parikh equivalent 1nfas must be at least n.

We now switch to the general case. We prove that for each input alphabet the state cost of the conversion of 1nfas into Parikh equivalent 1dfas is the same as for the unary alphabet.

In the proof we will also make use of the transformation of unary 1nfas into 1dfas given in Theorem 4.0.4.

Theorem 5.1.2. *For each n-state* 1nfa *over Σ, there exists a Parikh equivalent* 1dfa *with $e^{O(\sqrt{n \cdot \ln n})}$ states. Furthermore, this cost is tight.*

Proof. According to Lemma 3.4.1, from a given n-state 1nfa A with input alphabet $\Sigma = \{a_1, a_2, \ldots, a_m\}$, we build a 1nfa A_0 with $n(m+1)+1$ states that accepts the nonunary part of $L(A)$ and m n-state 1nfas $A_1, A_2, \ldots, A_m$ that accept the unary parts of $L(A)$. Using Theorem 4.0.4, for $i = 1, \ldots, m$, we convert A_i into an equivalent 1dfa A'_i with $e^{O(\sqrt{n \cdot \ln n})}$ states. We can assume that the state sets of the resulting automata are pairwise disjoint.

We define A_u that accepts $\{w \in L(A) \mid w \text{ is unary}\}$ consisting of one copy of each of these 1dfas and a new state q_s, which is its start state. In reading the first letter a_i of an input, A_u transits from q_s to the state q in the copy of A'_i if A'_i transits from its start state to q on a_i (such q is unique because A'_i is deterministic). These transitions from q_s do not introduce any nondeterminism because $A'_1, \ldots, A'_m$ are defined over pairwise distinct letters. After thus entering the copy, A_u merely simulates A'_i. The start state q_s should be also an accepting state if and only if $\varepsilon \in L(A'_i)$ for some $1 \leq i \leq m$. Being thus built, A_u accepts exactly all the unary words in $L(A)$ and contains at most $m \cdot e^{O(\sqrt{n \cdot \ln n})} + 1$ states.

On the other hand, for the nonunary part of $L(A)$, using Theorem 5.1.1, we convert A_0 into a Parikh equivalent 1dfa A_n with a number of states $r(n)$, polynomial in n. The standard product construction is applied to A_u

and A_n in order to build a 1dfa accepting $L(A_u) \cup L(A_n)$. The number of states of the 1dfa thus obtained is bounded by the product $e^{O(\sqrt{n \cdot \ln n})} \cdot r(n) = e^{O(\sqrt{n \cdot \ln n})} \cdot e^{O(\ln n)} = e^{O(\sqrt{n \cdot \ln n} + \ln n)}$, which is still bounded by $e^{O(\sqrt{n \cdot \ln n})}$.

Finally, we observe that by Theorem 4.0.4, in the unary case $e^{\Theta(\sqrt{n \cdot \ln n})}$ is the tight cost of the conversion from n-state 1nfas to 1dfas. This implies that the upper bound we obtained here cannot be reduced and so it is optimal.

This completes the proof of Theorem 5.1.2. $\square$

We conclude this section with some observations. We proved that for a fixed alphabet Σ, the state cost of the conversion of n-state 1nfas into Parikh equivalent 1dfas is polynomial in n, in the case each word in the accepted language is nonunary. The degree of this polynomial depends on m, the cardinality of the input alphabet.

For general regular languages, possibly containing unary words, the cost is superponynomial but subexponential in n. The optimality of this superponynomial cost is given by the “hard” case, on unary inputs. A closer inspection to our proofs shows that these costs are exponential in the size of the alphabet.

It could be pointed out here, that Parikh equivalent 1dfas can, in some sense, “capture” the Parikh characterization of regular languages polynomially, independently whether the languages do contain or do not contain unary words, as follows:

For the given regular language L (and its 1nfa), built over an m-letter alphabet $\Sigma = \{a_1, \ldots, a_m\}$, consider a new language $L' = \{a_0\} \cdot L$, where a_0 is a new symbol, built over an $(m+1)$-letter alphabet $\Sigma' = \{a_0, a_1, \ldots, a_m\}$. Clearly, except for $w = \varepsilon$, each $w \in L$ is mapped into a nonunary word $w' = a_0 w \in L'$. Moreover, if L is accepted by an n-state 1nfa, then $n+1$ states are sufficient for L', and hence the new 1nfa accepting L' can always be converted into a Parikh equivalent 1dfa with a polynomial number of states. It is also easy to see that the input w is mapped into a Parikh image $\boldsymbol{v} = (v_1, \ldots, v_m)$ if and only if w' is mapped into the Parikh image $\boldsymbol{v'} = (1, v_1, \ldots, v_m)$, that is, into an $(m+1)$-dimensional vector always having $\boldsymbol{v'}[1] = 1$. In this sense, the construction of Parikh equivalent 1dfa is never “hard”.

5.1.2 From cfgs to Parikh equivalent 1dfas

In this section we extend the results of Section 5.1.1 to the conversion of cfgs in Chomsky normal form into Parikh equivalent 1dfas. Actually, Theorem 5.1.1 will play an important role in order to obtain the main result of this section. The other important ingredient is the following result proven in 2011 by Esparza, Ganty, Kiefer and Luttenberger [EGKL11], which gives the cost of the conversion of Cnfgs into Parikh equivalent 1nfas.

Theorem 5.1.3 ([EGKL11])**.** *For each* Cnfg *with h variables, there exists a Parikh equivalent* 1nfa *with $\binom{2h+1}{h} = O(4^h)$ states.*

We point out that the upper bound in Theorem 5.1.3 does not depend on the cardinality of the input alphabet.

By combining Theorem 5.1.3 with the main result of the previous section, i.e., Theorem 5.1.2, we can immediately obtain a double exponential upper bound in h for the size of 1dfas Parikh equivalent to Cnfgs with h variables. However, we can do better. In fact, we show how to reduce the upper bound to a single exponential in a polynomial of h. We obtain this result by proceeding as in the case of finite automata: we split the language defined by given grammar into the unary and nonunary parts, we make separate conversions, and finally we combine the results.

As in Section 5.1.1, from now on let us fix the alphabet $\Sigma = \{a_1, a_2, \dots, a_m\}$. We start by considering the nonunary part. By combining Theorem 5.1.3 with Theorem 5.1.1 we obtain:

Theorem 5.1.4. *For each h-variable* Cnfg *with terminal alphabet Σ, generating a language none of whose words are unary, there exists a Parikh equivalent* 1dfa *with $2^{O(h)}$ states. Furthermore, this cost is tight.*

Proof. First, according to Theorem 5.1.3, we can transform the grammar into a Parikh equivalent 1nfa with $O(4^h)$ states. Then, using Theorem 5.1.1, we convert the resulting automaton into a Parikh equivalent 1dfa, with a number of states polynomial in $4^h = 2^{2h}$, hence exponential in h.

Even the bound given in Theorem 5.1.4, for languages consisting only of nonunary words, cannot be improved, by replacing the exponential in h by a slowly increasing function. This can be shown by adapting a standard argument from the unary case (e.g., [PSW02, Thm. 5]). For any integer $h \geq 3$, consider the grammar G with variables $A, B, A_0, A_1, \dots, A_{h-3}$, and productions

$$A \to a\,,\ B \to b\,,\ A_0 \to AB\,,\ A_j \to A_{j-1}A_{j-1}\,,\ \text{for } j = 1, ..., h-3\,.$$

An easy induction shows that, for $j = 1, \dots, h-3$, the only word which is generated from A_j is $(ab)^{2^j}$. Hence, by choosing A_{h-3} as start symbol, we have $L(G) = \{(ab)^H\}$, with $H = 2^{h-3}$. An immediate pumping argument shows that each 1dfa (or even 1nfa) with less than $2H+1$ states accepting a word of length $2H$, should also accept some words of length $< 2H$. Since $L(G)$ contains only the word $(ab)^H$, it turns out that each 1dfa accepting a language Parikh equivalent to $L(G)$ requires $2H+1$ states, namely a number exponential in h. □

Now, we switch to the general case. In the proof we will also make use of the transformations of unary Cnfgs into 1dfas given in Theorem 4.0.3.

Theorem 5.1.5. *For each h-variable* Cnfg *with terminal alphabet Σ, there exists a Parikh equivalent* 1dfa *with at most $2^{O(h^2)}$ states. Furthermore, this cost is tight.*

Proof. Let us denote the given Cnfg by $G = (V, \Sigma, P, S)$, where $|V| = h$.

In the case $m = 1$ (unary alphabet), one can employ Theorem 4.0.3. Note that, over a unary alphabet, two languages L_1, L_2 are Parikh equivalent if and only if they are equivalent. Hence, from now on we assume $m \geq 2$.

Let us give an outline of our construction first:

1. From G, we first create Cnfgs $G_0, G_1, \ldots, G_m$ such that G_0 generates the nonunary part of $L(G)$ and $G_1, G_2, \ldots, G_m$ generate the unary parts.

2. The grammars $G_1, G_2, \ldots, G_m$ are converted into respectively equivalent unary 1dfas $A_1, A_2, \ldots, A_m$. From these 1dfas, a 1dfa A_{unary} accepting the set of all unary words in $L(G)$ is constructed.

3. The grammar G_0 is converted into a Parikh equivalent 1dfa A_{non}.

4. Finally, from A_{unary} and A_{non}, a 1dfa that accepts the union of $L(A_{\text{unary}})$ and $L(A_{\text{non}})$ is obtained.

Observe that $L(A_{\text{unary}}) = \{w \in L(G) \mid w \text{ is unary}\}$ and $L(A_{\text{non}})$ is Parikh equivalent to $L(G_0) = \{w \in L(G) \mid w \text{ is not unary}\}$. Thus, the 1dfa which is finally constructed by this procedure is Parikh equivalent to the given grammar G.

We already have all the tools we need to implement each step in the above construction.

1. We can obtain grammars $G_0, G_1, \ldots, G_m$ according to Lemma 3.4.2. In particular, G_0 has $mh - m + 1$ variables, while each of the remaining grammars has h variables.

2. According to Theorem 4.0.3, for $i = 1, \ldots, m$, grammar G_i is converted into a 1dfa A_i with less than 2^{h^2} states. Using the same strategy presented in the proof of Theorem 5.1.2, from $A_1, \ldots, A_m$, we define A_{unary} consisting of one copy of each of these 1dfas and a new state q_s, which is its start state. Hence, the number of states of A_{unary} does not exceed $m2^{h^2} + 1$.

3. This step is done using Theorem 5.1.4. The number of the states of the resulting 1dfa A_{non} is exponential in the number of the variables of the grammar G_0 and, hence, exponential in h.

4. The final 1dfa can be obtained as the product of two automata A_{unary} and A_{non}. Considering the bounds obtained in Step 2 and 3 we conclude that the number of states in exponential in h^2.

We briefly discuss how the upper bounds depends on m, the alphabet size. Using the estimation of the cost of the conversions in Theorem 5.1.1 and observing that the grammar G_0 in the previous construction has less than mh variables, we can conclude that the automaton A_{non} has $2^{O(hm^4)}$ states. Hence, the number of the states of the resulting 1dfa of the above construction is $2^{O(h^2+hm^4)}$.

We observe that in [PSW02, Thm. 7] it was proved that there is a constant $c > 0$ such that for infinitely many $h > 0$ there exists a Cnfg with h variables generating a unary language such that each equivalent 1dfa requires at least 2^{ch^2} states. This implies that the upper bound in Theorem 5.1.5 cannot be improved. $\square$

We point out that in [LP12] it has been proved a result close to Theorem 5.1.5 in the case of Cnfgs generating *letter bounded languages*, i.e., subsets of $a_1^* a_2^* \cdots a_m^*$. In particular, an upper bound exponential in a polynomial in h has been obtained. However, the degree of the polynomial is, in turn, a polynomial in the size m of the alphabet. Here, in our Theorem 5.1.5, the degree is 2. Hence, it does not depend on m.

5.2 Conversions into Parikh equivalent 2dfas

In this section we study the conversions of 1nfas and Cnfgs into Parikh equivalent *two-way* deterministic automata. In the previous section, for the conversions into *one-way* deterministic automata, we observed that the unary parts are the most expensive. However, the cost of the conversions of unary 1nfas and Cnfgs into 2dfas are smaller than the costs for the corresponding conversions into 1dfas. We reduce the cost from an exponential in $\sqrt{n \ln n}$ to a polynomial in n and, from an exponential in a polynomial in h to an exponential in h, respectively. This allows us to prove that, in the general case, the cost of the conversions of 1nfas and Cnfgs into Parikh equivalent 2dfas are smaller than the cost of the corresponding conversions into 1dfas.

Let us start by presenting the following result, which derives from [Chr86, Thm. 6.2]:

Theorem 5.2.1. *For each n-state unary* 1nfa *there exists an equivalent halting* 2dfa *with $n^2 + 1$ states. Furthermore, this cost is tight.*

Proof. For the sake of completeness, we present a proof which is essentially the same given by Chrobak [Chr86, Thm. 6.2] where, however, the obtained upper bound was $O(n^2)$. Then we will explain why the big-O in the upper bound can be removed.

First of all, each n-state unary 1nfa A can be converted into an equivalent 1nfa A_c in a special form, which is known as *Chrobak normal form* [Chr86, Lemma 4.3], consisting of a deterministic path which starts from the initial state, and $k \geq 0$ disjoint deterministic cycles. The number of states in the path is $s = O(n^2)$, while the *total* number of states in the cycles is $r \leq n$. From the last state of the path there are k outgoing edges, each one of them reaching a fixed state on a different cycle. Hence, on each input of length $\ell \geq s$, the computation visits all the states on the initial path, until the last one where the *only* nondeterministic choice is taken, moving to one of the cycles, where the remaining part of the input is examined. However, if $\ell < s$ then computation ends in a state on the initial path, without reaching any loop. (In the special case $k = 0$ the accepted language is finite.)

A 2dfa B can simulate the 1nfa A_c in Chrobak normal form, traversing the input word at most $k + 1$ times. In the first traversal, the automaton checks whether or not the input length is less than s. If this is the case, then the automaton accepts or rejects according to the corresponding state on the initial path of A_c. Otherwise, it moves to the right endmarker. This part can be implemented with $s + 1$ states (s states for the simulation of the initial path, plus one more state to move to the right endmarker). From the right endmarker, the automaton traverses the input leftward, by simulating the first cycle of A_c from a suitable state (which is fixed, only depending on s and on the cycle length). If the left endmarker is reached in a state which simulates a final state in the cycle then the automaton B moves to the final state q_f and accepts, otherwise it traverses the input rightward, simulating the 2nd cycle of A_c, and so on. Hence, in the $(i + 1)$th traversal of the input, $1 \leq i \leq k$, the ith cycle is simulated. So the number of states used to simulate the cycles is equal to the total number of states in the cycles, namely r. Considering the final state q_f, we conclude that B can be implemented with $s + r + 2 = O(n^2)$ states.

Finally, we point out that finer estimations for the number of the states on the initial path and in the loops of A_c have been found. In [Gef07], it was proved that the number of s of the states in the initial path is bounded by $n^2 - 2$ and the sum r of the numbers of the states in the cycles is bounded by $n - 1$. Actually, there is an exception: if the given 1nfa A is just one cycle of n states then A is already a 1dfa. If it is minimal, then in any equivalent 1nfa we cannot have a cycle with less than n states which is useful to accept some input. However, in this degenerate case, Theorem 5.2.1 is trivially true, without making use of the Chrobak normal form.

The first bound has been further reduced in [Gaw11] to $s \leq n^2 - n$. This allows us to conclude that the 2dfa B can be obtained with at most $n^2 + 1$ states.

The upper bound given in Theorem 5.2.1 is asymptotically tight. As proven in [Chr86, Thm. 6.3], for each integer n there exists an n-state unary 1nfa such that any equivalent 2dfa requires $\Omega(n^2)$ states. $\square$

By combining Theorem 5.2.1 with the bound for the transformation of unary Cnfgs into 1nfas given in Theorem 4.0.3, we immediately obtain the following bound.

Theorem 5.2.2. *For each h-variable unary* Cnfg *there exists an equivalent halting* 2dfa *with at most* $(2^{2h-1}+1)^2+1$ *states. Moreover, this cost is tight.*

We now have the tools for studying the conversions of 1nfas and cfgs into Parikh equivalent 2dfas. Let us start with the first conversion.

5.2.1 From 1nfas to Parikh equivalent 2dfas

Theorem 5.2.3. *For each n-state* 1nfa *there exists a Parikh equivalent* 2dfa *with a number of states polynomial in n. Furthermore, this cost is tight.*

Proof. We use the same technique as in the proof of Theorem 5.1.2, by splitting the language accepted by the given 1nfa A into its unary and nonunary parts, as explained in Lemma 3.4.1. Each unary part is accepted by a 1nfa with n states. According to Theorem 5.2.1, this gives us m 2dfas $B_1, B_2, \ldots, B_m$, accepting the unary parts, each one of them has at most n^2+1 states, where m is the cardinality of the input alphabet $\Sigma = \{a_1, a_2, \ldots, a_m\}$.

For the nonunary part we have a 1nfa with $n(m+1)+1$ states and, according to Theorem 5.1.1, a Parikh equivalent 1dfa B_0 with a number of states polynomial in $n(m+1)+1$ and, hence, in n.

Finally, as explained in Section 3.4.2, we can build a 2dfa B such that $L(B) = L(B_0) \cup L(B_1) \cup \cdots \cup L(B_m)$. Hence, B is Parikh equivalent to the given 1nfa A and its number of states is polynomial in n.

By making the same considerations as in Theorem 5.1.1 we can obtain an $O(m^3)$ bound for the degree of the polynomial.

Finally, we observe that by Theorem 5.2.1, in the unary case $\Theta(n^2)$ is the tight cost of the conversion from n-state 1nfas to 2dfas. This implies that the upper bound we obtained here cannot be reduced. □

5.2.2 From cfgs to Parikh equivalent 2dfas

Now, we consider the conversion of cfgs.

Theorem 5.2.4. *For each h-variable* Cnfg *there exists a Parikh equivalent* 2dfa *with* $2^{O(h)}$ *states. Furthermore, this cost is tight.*

Proof. Even in this case, the construction is obtained by adapting the corresponding conversion into 1dfas (Theorem 5.1.5). In particular, the construction uses the same steps 1-4 given in that proof, with some modifications in steps 2 and 4, which are replaced by the following ones:

2'. The grammars $G_1, G_2, \ldots, G_m$ are converted into respectively equivalent unary 2dfas $A'_1, A'_2, \ldots, A'_m$.

4'. Finally, from $A_{\text{non}}, A'_1, A'_2, \ldots, A'_m$, a 2dfa that accepting the language $L(A_{\text{non}}) \cup L(A'_1) \cup L(A'_2) \cup \cdots \cup L(A'_m)$ is obtained.

Clearly, the 2dfas resulting from this procedure is Parikh equivalent to the original grammar G. The costs of steps 1 and 3 have been discussed in the proof of Theorem 5.1.5. For the remaining steps:

2'. According to Theorem 5.2.2, for $i = 1, \ldots, m$, the 2dfa A'_i has at most $(2^{2h-1} + 1)^2 + 1$ states.

4'. We use the construction presented at the end of Section 3.4.2, to obtain a 2dfa whose number of states is the sum of the number of the states of $A_{\text{non}}, A'_1, A'_2, \ldots, A'_m$, hence $2^{O(h)}$. Explicitly mentioning the dependency on the alphabet size m, we can give a $2^{O(hm^4)}$ bound. This derives from the size of the automaton A_{non} (cf. Theorem 5.1.5).

Finally, we observe that by Theorem 5.2.2, in the unary case $2^{\Theta(h)}$ is the tight cost of the conversion from h-variable Cnfgs to 2dfas. This implies that the upper bound we obtained here cannot be reduced. □

Chapter 6

Operational state complexity under Parikh equivalence

La Primavera - Allegro

Il cimento dell'armonia e dell'inventione
Antonio Vivaldi

In this chapter, we investigate the state complexity of operations between regular languages under Parikh equivalence. Operations include concatenation, Kleene star, reversal, shuffle, projection, union, intersection, and complementation.

The problems we will work on in this chapter can be formalized in the following general form:

Problem 6.0.1. *For* 1dfa*s A and B of n_1 and n_2 states, respectively, solve the following problems:*

1. *For a unary operation f, how small can we make a* 1dfa *M that is Parikh equivalent to $f(L(A))$?*
2. *For a binary operation g, how small can we make a* 1dfa *M that is Parikh equivalent to $g(L(A), L(B))$?*

In other words, given two 1dfas A and B of n_1 and n_2 states, we want to build a 1dfa that accepts a language Parikh equivalent to L, where L is the result of an operation between the languages accepted by A and B.

6.1 Union, intersection and complement

The state complexity of union and intersection is in the low order n_1n_2 even in the conventional sense over both unary and nonunary alphabets.

Moreover, it is known to be tight already over a unary alphabet [Yu00]. Similar considerations hold for the complement. The next result hence follows.

Proposition 6.1.1. *Given two* 1dfa*s* A *and* B *of* n_1 *and* n_2 *states, respectively, there exist two* 1dfa*s of* n_1n_2 *states that accept languages (Parikh) equivalent to* $L(A) \cup L(B)$ *and to* $L(A) \cap L(B)$*, respectively, and a* 1dfa *of* n_1 *states that accepts a language (Parikh) equivalent to the complement of* $L(A)$*. These bounds are tight.*

6.2 Concatenation, shuffle and star

Unlike union or intersection, concatenation is known to cost 1dfas an exponential number of states. In fact, the number of states which is necessary and sufficient in the worst case for a 1dfa to accept the concatenation of an n_1-state 1dfa language and an n_2-state 1dfa language over a binary alphabet is $(2n_1 - 1)2^{n_2-1}$ [YZS94]; over a unary alphabet, the cost decreases down to n_1n_2 [Yu00].

We will show that under Parikh equivalence, the state complexity of concatenation decreases down to polynomial in n_1 and n_2 over an arbitrary alphabet. The Parikh equivalent conversion of 1nfas to 1dfas in Theorem 5.1.1 makes a great contribution to this purpose.

For concatenation, we could first build a 1nfa with $n_1 + n_2$ states and then according to the superpolynomial conversion of 1nfas into Parikh equivalent 1dfas presented in Theorem 5.1.2 we could convert it into a Parikh equivalent 1dfa with a superpolynomial numbers of states. In order to avoid a superpolynomial blowup in the number of states caused by this conversion when being applied to the unary part of a 1nfa, we give an *ad hoc* constructions below.

Problem 6.2.1. *Given two arbitrary* 1dfa*s* A *and* B *of* n_1 *and* n_2 *states, respectively, how many states are sufficient and necessary in the worst case (as a function of* n_1 *and* n_2*) for a* 1dfa *to accept a language Parikh equivalent to the concatenation of the languages accepted by* A *and* B*?*

Let us analyze the case $\Sigma = \{a, b\}$, that is, $m = 2$. Given two 1dfas A and B, let $L = L(A)L(B)$. The idea is to split L into unary and nonunary parts. To do so we proceed as follows:

- Let 1dfas A_1, A_2, A_0 such that $L(A_1) = L(A) \cap \{a\}^*$, $L(A_2) = L(A) \cap \{b\}^*$ be unary parts and $L(A_0) = L(A) \setminus (L(A_1) \cup L(A_2))$ be nonunary part of A. This is possible by Lemma 3.4.1. In particular, automata A_1, A_2 are unary 1dfas of n_1 states.
- Similarly, let 1dfas B_1, B_2, B_0, such that $L(B_1) = L(B) \cap \{a\}^*$, $L(B_2) = L(B) \cap \{b\}^*$ be unary parts and $L(B_0) = L(B) \setminus (L(B_1) \cup L(B_2))$ be nonunary part of B. In this case B_1, B_2 are unary 1dfas of n_2 states.

Solving the concatenation between $L(A)$ and $L(B)$, we observe in the Equation (6.1) that $L(A_1)L(B_1) = L \cap \{a\}^*$ and $L(A_2)L(B_2) = L \cap \{b\}^*$ are the only unary languages, while the others languages are nonunary.

$$\begin{aligned} L(A)L(B) &= (L(A_0) \cup L(A_1) \cup L(A_2))(L(B_0) \cup L(B_1) \cup L(B_2)) \\ &= L(A_0)L(B_0) \cup L(A_0)L(B_1) \cup L(A_0)L(B_2) \\ &\cup L(A_1)L(B_0) \cup \mathbf{L(A_1)L(B_1)} \cup L(A_1)L(B_2) \\ &\cup L(A_2)L(B_0) \cup L(A_2)L(B_1) \cup \mathbf{L(A_2)L(B_2)} \end{aligned} \tag{6.1}$$

Let L_0 be the nonunary part of L and $L_1 = L(A_1)L(B_1)$, $L_2 = L(A_2)L(B_2)$ be unary parts of L.

- For the nonunary part of L, let us denote by N the $(n_1 + n_2)$-state 1nfa obtained by applying the standard construction for the product to A and B. From N we derive a 1nfa accepting L_0 and we convert it according to Lemma 3.4.1 into a 1nfa N_0 with $n_0 = (n_1+n_2)(m+1)+1$ states. Using Theorem 5.1.1 there exists a Parikh equivalent 1dfa $\widehat{N}_0$ with a polynomial number of states in n_0.

- For the unary parts of L we can use a known result presented in [PS02] to build unary 1dfas M_1, M_2 accepting the unary languages L_1, L_2 both with at most $n_1 n_2$ states.

Over an arbitrary alphabet, we obtain the following result.

Theorem 6.2.1. *Given two* 1dfa*s A and B of n_1 and n_2 states, respectively, there exists a* 1dfa *of polynomial number of states in n_1 and n_2 that is Parikh equivalent to $L(A)L(B)$.*

Proof. Let $L = L(A)L(B)$. For each $a_i \in \Sigma$, let us denote by L_i the language $L \cap \{a_i\}^*$ and by L_0 the language $L \setminus (\bigcup_{i=1}^{m} L_i)$, namely the nonunary part of L.

Let $A_1, \ldots, A_m$ be the unary 1dfas of n_1 states obtained by applying Lemma 3.4.1 to the 1dfa A. Let $B_1, \ldots, B_m$ be the unary 1dfas of n_2 states thus obtained from B. Let us denote by M the (n_1+n_2)-state 1nfa obtained by applying the standard construction for the product to A and B. Then we proceed as follows:

- From M we derive a 1nfa accepting L_0 with $n_0 = (n_1 + n_2)(m + 1) + 1$ states by Lemma 3.4.1 and we convert it according to Theorem 5.1.1 into a polynomial size Parikh equivalent 1dfa M_0 with at most $O({n_0}^{3m^3+6m^2} m^{m^3/2+m^2+2m+5})$ states. It consists of $O((2(n_1 + n_2))^{3m^3+6m^2} m^{7m^3/2+7m^2+2m+5})$ states.

- For $i = 1, \ldots, m$, from A_i and B_i we obtain a unary 1dfa M_i accepting the language L_i. According to results in [PS02] the number of states of M_i can be bounded by $n_1 n_2$.
- We build a 1dfa M' with at most $1 + mn_1n_2$ states accepting all the unary words in L, namely the set $\bigcup_{i=1}^{m} L_i$. This consists of an initial state q_0 and one copy of each automaton M_i obtained in the previous step. A transition from q_0 on a_i to an appropriate state of M_i simulates the transition from the initial state of M_i.
- Finally, applying the standard construction for the union from 1dfas M_0 and M' we get a 1dfa of polynomial number of states in n_1 and n_2 that accepts a language which is Parikh equivalent to $L = L(A)L(B)$. It consists of $O(n_1n_2(2(n_1+n_2))^{3m^3+6m^2} m^{7m^3/2+7m^2+2m+6})$ states.

$\square$

We have two remarks concerning Theorem 6.2.1. First, the degree in the upper bound given by the expression written at the end of the proof could be reduced under the assumption that $L(A)L(B)$ does not contain unary strings. Anyway, the bound always remains a polynomial asymptotically. Furthermore, we observe that in the unary case n_1n_2 [Yu00] is the tight cost for the concatenation of an n_1-state 1dfa language and an n_2-state 1dfa language. This implies that the upper bound we obtained here cannot be reduced below the polynomial n_1n_2.

Now we address the state complexity of shuffle under Parikh equivalence. Let A and B be 1dfas of n_1 and n_2 states, respectively. In the conventional sense, shuffle involves the exponential cost $2^{n_1n_2}-1$ and this bound is tight [CSY02]. In contrast, we can construct a 1dfa of polynomial number of states in n_1 and n_2 that accepts a language Parikh equivalent to the shuffle of $L(A)$ and $L(B)$, and in fact, the 1dfa we engineered in Theorem 6.2.1 for concatenation is such a 1dfa. This is because the Parikh image of the shuffle of two languages is equal to that of their concatenation.

Corollary 6.2.2. *Given two* 1dfa*s A and B of n_1 and n_2 states, respectively, there exists a* 1dfa *of polynomial number of states in n_1 and n_2 that is Parikh equivalent to the shuffle of $L(A)$ and $L(B)$.*

Now we analyze the state complexity of star under Parikh equivalence. Like concatenation, star is known to cost 1dfas an exponential number of states. In fact, the number of states which is necessary and sufficient in the worst case for a 1dfa to accept the star of an n-state 1dfa language over a binary alphabet is $2^{n-1} + 2^{n-2}$, whereas it is $(n-1)^2 + 1$ over a unary alphabet [YZS94].

Let us develop the above conversion of polynomial overhead for star. We will show that under Parikh equivalence, the state complexity of star decreases down to polynomial in n_1 and n_2 over an arbitrary alphabet.

Theorem 6.2.3. *Given a* 1dfa *A of n states, there exists a* 1dfa *of polynomial number of states in n that is Parikh equivalent to $L(A)^*$. Moreover, this cost is tight.*

Proof. The idea is to split $L(A)^*$ into unary and nonunary parts.

Lemma 3.4.1 yields, from the given 1dfa A, 1dfas $A_1, \ldots, A_m$ (of n states) for the unary parts. From them, we can construct 1dfas $M_1, \ldots, M_m$ of $(n-1)^2+1$ states accepting $L(A_1)^*, \ldots, L(A_m)^*$, respectively [YZS94]. We further combine them into a 1dfa M_{unary} of $m((n-1)^2+1)+1$ states that accepts $L(A_1)^* \cup \cdots \cup L(A_m)^*$.

From A, we also construct an n-state 1nfa recognizing $L(A)^*$, extract its nonunary part by Lemma 3.4.1 (with $n(m+1)+1$ states), and convert it into a Parikh equivalent 1dfa M_0 using Theorem 5.1.1. The resulting 1dfa M_0 consists of $O((2n)^{3m^3+6m^2} m^{7m^3/2+7m^2+2m+5})$ states.

Applying the standard construction for the union from 1dfas M_0 and M_{unary} we get a polynomial size 1dfa that accepts a language Parikh equivalent to $L(A)^*$. The 1dfa consists of $O((2n)^{3m^3+6m^2+1} n m^{7m^3/2+7m^2+2m+6})$ states.

Finally, we can observe that by the unary case, $(n-1)^2+1$ is the tight cost for the star of an n-state 1dfa language [YZS94]. This implies that the upper bound we obtained here cannot be reduced. ☐

6.3 Reversal and projection

Reversal is also expensive for 1dfas. In fact, the tight bound 2^n is known for reversal [YZS94]. Under Parikh equivalence, however, nothing need be said since Parikh image is invariant under this operation.

It is easy to see that projection preserves regularity. However, transforming a 1dfa A of n states into a 1dfa for the projection can require a number of states that is exponential in n [JM12]. Even in this case, the bound can be reduced if we want to obtain a Parikh equivalent 1dfa: from A we can obtain a 1nfa of n states for the projection, and then we can transform it into a Parikh equivalent 1dfa of $e^{O(\sqrt{n \cdot \ln n})}$ states. By using a projection over a unary alphabet we can show that this bound cannot be reduced.

Theorem 6.3.1. *Let A be a* 1dfa *with n states. Then:*

1. *There exists a* 1dfa *with n states that accepts a language Parikh equivalent to the reversal of $L(A)$.*

2. *There exists a* 1dfa *with $e^{O(\sqrt{n \cdot \ln n})}$ states that accepts a language Parikh equivalent to the projection $P_{\Sigma_0}(L(A))$ of $L(A)$ over $\Sigma_0 \subseteq \Sigma$.*

Moreover, these bounds are asymptotically tight.

6.4 Intersection and complement, revisited

We consider one more time the intersection and the complement. In fact, the noncommutativity of those operations with the Parikh mapping brings us a second problem of interest. The noncommutativity in the case of intersection is illustrated in the inequality $\psi(a^+b^+ \cap b^+a^+) \neq \psi(a^+b^+) \cap \psi(b^+a^+)$; the left-hand side is the empty set, while the right-hand side is the linear set $\mathbb{N} \times \mathbb{N}$. In the case of complement the reader may consider the language $(ab)^*$.

Note that each of the other operations examined so far is either commutative with the Parikh mapping (i.e., union and projection) or not defined naturally over the set of nonnegative integer vectors (i.e., concatenation, star, shuffle, and reversal). The problem of interest asks: given two 1dfas A and B of n_1 and n_2 states, respectively, how small can we make a 1dfa whose Parikh image is equal to $\psi(L(A)) \cap \psi(L(B))$? We can formulate a similar problem in the case of the complement. The fact that the Parikh image of a language accepted by an 1nfa is semilinear and the closure property of semilinear sets under intersection and complement [GS64] makes these problems meaningful. Here, we solve the problem for intersection, leaving the one for complement for future investigations.

Over a unary alphabet, the problem is degenerated into the problem addressed in Proposition 6.1.1 because over such an alphabet, intersection commutes with the Parikh mapping. Therefore, in the following, we examine the problem over a nonunary alphabet, and solve it by showing that a polynomial number of states in n_1 and n_2 are sufficient. The proof consists of revisiting the Ginsburg and Spanier's proof [GS64, Thm. 6.1] of the closure property of semilinear sets under intersection with a careful analysis of the size of the resulting semilinear set.

Let us present some notation and results useful in the proof. Given $C, P \subseteq \mathbb{N}^k$ for some $k \geq 1$, let $L(C; P)$ be the set of all $\boldsymbol{w} \in \mathbb{N}^k$ which can be represented in the form $\boldsymbol{w} = \boldsymbol{w}_0 + \boldsymbol{w}_1 + \cdots + \boldsymbol{w}_\ell$ with $\boldsymbol{w}_0 \in C$ and $\boldsymbol{w}_1, \ldots, \boldsymbol{w}_\ell \in P$ for some $\ell \geq 0$.

Let A be a $k \times \ell$ matrix with entries in $\mathbb{Z}$ and $k \leq \ell$, and let $\boldsymbol{b} \in \mathbb{Z}^k$. Consider a system of linear Diophantine equations

$$\mathrm{A}\boldsymbol{x} = \boldsymbol{b}. \tag{6.2}$$

Let $S_{\min}(\mathrm{A}, \boldsymbol{b})$ be the set of minimal nonnegative integer solutions to (6.2), where the minimality is with respect to the component-wise comparison. It is well known that $S_{\min}(\mathrm{A}, \boldsymbol{b})$ is finite. Hence, we let $S_{\min}(\mathrm{A}, \boldsymbol{b}) = \{s_1, s_2, \ldots, s_r\}$ for some $r \geq 1$ and $s_1, \ldots, s_r \in \mathbb{N}^\ell$. Define $||S_{\min}(\mathrm{A}, \boldsymbol{b})|| = \max_{1 \leq i \leq r} ||s_i||_\infty$, where $||s_i||_\infty$ refers to the maximum norm. Huynh bounded $||S_{\min}(\mathrm{A}, \boldsymbol{b})||$ as follows (see Theorem 2.6 and Corollary 2.7 in [Huy80]).

Lemma 6.4.1 ([Huy80]). *Let α be the rank of* A *and M be the maximum of the absolute values of the $\alpha \times \alpha$ subdeterminants of the extended matrix* $(\mathrm{A}; \boldsymbol{b})$. *Then $||S_{\min}(\mathrm{A}, \boldsymbol{b})|| \leq (\ell+1)M$. Thus, $r \leq ((\ell+1)M+1)^{\ell}$.*

Theorem 6.4.1. *Given two* 1dfa*s A and B of n_1 and n_2 states, respectively, there exists a* 1dfa *of polynomial number of states in n_1 and n_2 whose Parikh image is equal to $\psi(L(A)) \cap \psi(L(B))$.*

Proof. As usual, we begin with converting the given 1dfas A and B into $2(m+1)$ 1dfas $A_0, A_1, \ldots, A_m, B_0, B_1, \ldots, B_m$ according to Lemma 3.4.1. The nonunary 1dfas A_0 and B_0 contain $n_1(m+1)+1$ and $n_2(m+1)+1$ states, respectively, while the other unary ones A_i and B_i contain only n_1 and n_2 states for $1 \leq i \leq m$, respectively. It is clear from the definition of these automata that

$$\psi(L(A)) \cap \psi(L(B)) = \Big(\psi(L(A_0)) \cap \psi(L(B_0))\Big) \cup \bigcup_{1 \leq i \leq m} \Big(\psi(L(A_i)) \cap \psi(L(B_i))\Big).$$

As observed before the proof, for any $1 \leq i \leq m$, we can construct a 1dfa M_i of $n_1 n_2$ states such that $\psi(L(M_i)) = \psi(L(A_i)) \cap \psi(L(B_i))$. We combine them into one 1dfa M_{unary} with mn_1n_2+1 states that accepts $\bigcup_{i=1}^{m} L(M_i)$.

What we have to consider now is the intersection between the Parikh images of the nonunary 1dfas A_0 and B_0. In order to simplify the notation below, let $n = \max\{n_1, n_2\}(m+1)+1$. Then, A_0 and B_0 consist of at most n states each. Lemma 4.0.2 implies that the Parikh image $\psi(L(A_0))$ can be represented as:

$$\psi(L(A_0)) = Y_A \cup \bigcup_{i \in I} Z_{A,i},$$

where $Y_A \subseteq \{0, 1, \ldots, p(n)\}^m$, I is a set of at most $p(n)$ indices, and for each $i \in I$, $Z_{A,i} \subseteq \mathbb{N}^m$ is a linear set whose offset is in $\{0, 1, \ldots, p(n)\}^m$ and whose generators are linearly independent vectors in $\{0, 1, \ldots, n\}^m$. The Parikh image of $L(B_0)$ also admits an analogous representation with Y_B, J, and $Z_{B,j}$ $(j \in J)$. The intersection $\psi(L(A_0)) \cap \psi(L(B_0))$ can be expressed as:

$$\Big(Y_A \cap \psi(L(B_0))\Big) \cup \left(\bigcup_{i \in I} Z_{A,i} \cap Y_B\right) \cup \left(\bigcup_{i \in I} Z_{A,i} \cap \bigcup_{j \in J} Z_{B,j}\right).$$

Since both Y_A and Y_B are finite, the first two terms are finite, and can be computed even by hand. Let $P_Y = \Big(Y_A \cap \psi(L(B_0))\Big) \cup \left(\bigcup_{i \in I} Z_{A,i} \cap Y_B\right)$, that is, their union. Note that $P_Y \subseteq Y_A \cup Y_B \subseteq \{0, 1, \ldots, p(n)\}^m$. We can construct a 1dfa M_Y of $O(p(n)^m)$ states that accepts the bounded language $\{a_1^{i_1} a_2^{i_2} \cdots a_m^{i_m} \mid (i_1, \ldots, i_m) \in P_Y\}$. It is clear that $\psi(L(M_Y)) = P_Y$.

Now we shift our attention to the third term $\Big(\bigcup_{i \in I} Z_{A,i} \cap \bigcup_{j \in J} Z_{B,j}\Big)$. What we actually do is to construct a 1dfa $B_{i,j}$ whose Parikh image is equal to $Z_{A,i} \cap Z_{B,j}$, for each $i \in I$ and $j \in J$. According to Lemma 4.0.2, for some $p, q \leq m$, we can let

$$\begin{aligned} Z_{A,i} &= \{\boldsymbol{u}_0 + n_1\boldsymbol{u}_1 + \cdots + n_p\boldsymbol{u}_p \mid n_1, \ldots, n_p \in \mathbb{N}\}, \\ Z_{B,j} &= \{\boldsymbol{v}_0 + m_1\boldsymbol{v}_1 + \cdots + m_q\boldsymbol{v}_q \mid m_1, \ldots, m_q \in \mathbb{N}\}, \end{aligned}$$

where $\boldsymbol{u}_0, \boldsymbol{v}_0 \in \{0, 1, \ldots, p(n)\}^m$ and $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_p, \boldsymbol{v}_1, \ldots, \boldsymbol{v}_q \in \{0, 1, \ldots, n\}^m$.

Let

$$\begin{aligned} X_C &= \left\{ (n_1, \ldots, n_p, m_1, \ldots, m_q) \;\middle|\; \boldsymbol{u}_0 + \sum_{i=1}^{p} n_i\boldsymbol{u}_i = \boldsymbol{v}_0 + \sum_{j=1}^{q} m_j\boldsymbol{v}_j \right\} \\ X_P &= \left\{ (n_1, \ldots, n_p, m_1, \ldots, m_q) \;\middle|\; \sum_{i=1}^{p} n_i\boldsymbol{u}_i = \sum_{j=1}^{q} m_j\boldsymbol{v}_j \right\} \end{aligned}$$

Let $\tau : \mathbb{N}^{p+q} \to \mathbb{N}^m$ be a function defined as $\tau((n_1, \ldots, n_p, m_1, \ldots, m_q)) = \sum_{i=1}^{p} n_i\boldsymbol{u}_i$, and we generalize this function over a set $X \subseteq \mathbb{N}^{p+q}$ as $\tau(X) = \{\tau(\vec{x}) \mid \vec{x} \in X\}$. Then

$$Z_{A,i} \cap Z_{B,j} = \{\boldsymbol{u}_0 + \boldsymbol{w} \mid \boldsymbol{w} \in \tau(X_C)\}. \tag{6.3}$$

Note that if X_C is semilinear, then so is $\tau(X_C)$ [GS64, Lemma 6.3]. Ginsburg and Spanier proved that X_C is semilinear [GS64]. More strongly, they gave a representation of X_C as $L(C; P)$, where C and P are the set of minimal elements of X_C and $X_P \setminus \{\mathbf{0}^{p+q}\}$, respectively. Lemma 6.4.1 gives the following bounds:

$$\begin{aligned} ||C|| &\leq (p + q + 1)m!n^{m-1}p(n), \\ ||P|| &\leq (p + q + 1)m!n^m. \end{aligned}$$

They bound the cardinality of the sets C and P as follows:

$$\begin{aligned} |C| &\leq ((p + q + 1)m!n^{m-1}p(n) + 1)^m, \\ |P| &\leq ((p + q + 1)m!n^m + 1)^m. \end{aligned}$$

Using $\tau(X_C) = \tau(L(C; P)) = L(\tau(C); \tau(P))$ (see the proof of Lemma 6.3 in [GS64]), we rewrite (6.3) as

$$Z_{A,i} \cap Z_{B,j} = \vec{u}_0 + L(\tau(C); \tau(P)).$$

Note that $|\tau(C)| \leq |C|$, $|\tau(P)| \leq |P|$, and we have

$$\begin{aligned} ||\tau(C)|| &\leq nm||C|| &&\leq m(p + q + 1)m!n^m p(n) \\ ||\tau(P)|| &\leq nm||P|| &&\leq m(p + q + 1)m!n^{m+1}. \end{aligned}$$

We construct an 1nfa whose Parikh image is equal to this semilinear set. Specifically, it is to accept the language $\{f(\vec{u}_0)f(\vec{v})f(\vec{w}) \mid \vec{v} \in \tau(C), \vec{w} \in$

$L(\vec{0}; \tau(P))\}$, where $\vec{0}$ is the zero vector and $f : \mathbb{N}^m \to \Sigma^*$ is defined as: $f((x_1, x_2, \ldots, x_m)) = a_1^{x_1} a_2^{x_2} \cdots a_m^{x_m}$. It first recognize $f(\vec{u}_0)$ using $mp(n) + 1$ states (recall that any coordinate of $\vec{u}_0$ is at most $p(n)$). Then it recognizes $f(\vec{v})$ for some $\vec{v}$ nondeterministically chosen from $\tau(C)$, using $m||\tau(C)||\cdot|\tau(C)|$ states. Finally it recognizes $f(\vec{w})$ for some $\vec{w} \in L(\vec{0}; \tau(P))$ using $m||\tau(P)|| \cdot |\tau(P)|$ states. In total, it consists of $O(m^2(2m+1)^2(m!)^2 n^{2m-1} p(n)^2)$ states. Now that we have at most $p(n)^2$ 1nfas $M_{0,0}, \ldots, M_{|I|-1,|J|-1}$ of such size. The standard construction combines them into an 1nfa with $O(m^2(2m+1)^2(m!)^2 n^{2m-1} p(n)^4)$ states and it is nonunary. The nonunarity allows Theorem 5.1.1 to convert this 1nfa into a Parikh equivalent 1dfa M_0 with $O(n^{(2m-1)(3m^3+6m^2)} p(n)^{2(3m^3+6m^2)})$ states.

Now that the standard construction for union combines this, M_Y, and M_{unary} into the 1dfa with $O(n^{(2m-1)(3m^3+6m^2)+2} p(n)^{2(3m^3+6m^2)+m})$ states, and its Parikh image is equal to $\psi(L(A)) \cap \psi(L(B))$. □

We conclude this section with some short considerations concerning the complement. It is well known that any regular language and its complement are accepted by 1dfas with the same number of states. Hence, this gives a trivial bound even under Parikh equivalence. However, as for the intersection, the complement does not commute with the Parikh mapping (consider, e.g., the language $(ab)^*$). Furthermore, the semilinear sets are closed under complementation [GS64]. Hence the problem arises of stating how small we can make a 1dfa accepting a language whose Parikh image is the complement of the Parikh image of the language accepted by a given 1dfa. We leave this problem for future investigations.

Chapter 7

Conclusion and future works

Hallelujah!

Messiah
George Frideric Handel

In this final chapter we briefly summarize the results shown in this thesis and we outline some possible future extensions of the research in the field of Parikh equivalence.

7.1 Conclusion

We proved that the state cost of the conversion of n-state 1nfas into Parikh equivalent 1dfas is $e^{\Theta(\sqrt{n \cdot \ln n})}$ (Theorem 5.1.2). This is the same cost of the conversion of unary 1nfas into equivalent 1dfas. Since in the unary case Parikh equivalence is just standard equivalence, this result can be seen as a generalization of the Chrobak conversion [Chr86] to the nonunary case.

More surprisingly, such a cost is due to the unary part of the languages. In fact, as shown in Theorem 5.1.1, for each n-state unary 1nfa accepting a language which does not contain any unary word, there exists a Parikh equivalent 1dfa with polynomially many states. Hence, while for the transformation from 1nfas to equivalent 1dfas, we need at least two different symbols to prove the exponential gap from n to 2^n states and we have a smaller gap in the unary case, for Parikh equivalence the worst case is due to unary words. Even in the proof of our result for cfgs (Theorem 5.1.5), the separation between the unary and nonunary parts was crucial. Also in this case, it turns out that the most expensive part is the unary one.

On the other hand, in our conversions into Parikh equivalent 2dfas, the most expensive part turns out to be the nonunary one.

The state costs of the conversions from 1nfas and cfgs into Parikh equivalent 1dfas and 2dfas are summarized in Table 7.1 and Table 7.2, respectively.

Tables also show the lower bounds that come from unary cases, when the alphabet is unary.

	Upper bound	Lower bound
n-state 1nfa	$e^{O(\sqrt{n \cdot \ln n})}$ Thm. 5.1.2	$e^{\Theta(\sqrt{n \cdot \ln n})}$ Thm. 4.0.4
h-variable Cnfg	$2^{O(h^2)}$ Thm. 5.1.5	$2^{\Theta(h^2)}$ Thm. 4.0.3

Table 7.1: Conversions from 1nfas and cfgs into Parikh equivalent 1dfas.

	Upper bound	Lower bound
n-state 1nfa	$poly(n)$ Thm. 5.2.3	$\Theta(n^2)$ Thm. 5.2.1
h-variable Cnfg	$2^{O(h)}$ Thm. 5.2.4	$2^{\Theta(h)}$ Thm. 5.2.2

Table 7.2: Conversions from 1nfas and cfgs into Parikh equivalent 2dfas.

In Chapter 6 we reformulated, under Parikh equivalence, some classical questions on the state complexity of operations on regular languages and on 1dfas. For union, intersection, complement, concatenation, Kleene star, shuffle and reversal operations, we proved a polynomial state complexity over any fixed alphabet, in contrast to the intrinsic exponential state complexity of some of these operations under standard equivalence. For the case of projection we proved a superpolynomial state complexity, which is lower than the exponential one of the corresponding classical operations.

Table 7.3 shows the state complexity of operations on regular languages treated under Parikh equivalence and standard equivalence. We assume that L_1 is an n_1-state 1dfa language and L_2 is an n_2-state 1dfa language defined over an m-letter alphabet, and $n_1, n_2 > 1$. The lower bounds of the operations under Parikh equivalence coincide with the state costs of the operations under standard equivalence when the alphabet is unary ($m = 1$).

Concerning intersection, we observed that this operation does not commute with Parikh image. However, semilinear sets are closed under intersection [GS64]. We proved that for each two 1dfas A and B it is possible to obtain a 1dfa with a polynomial number of states, whose Parikh image is the intersection of the Parikh images of the languages accepted by A and B (Theorem 6.4.1).

Operation	Parikh equivalence	Standard equivalence	
		$m = 1$	$m \geq 2$
$L_1 \cup L_2$	$n_1 n_2$	$n_1 n_2$	$n_1 n_2$
$L_1 \cap L_2$	$n_1 n_2$	$n_1 n_2$	$n_1 n_2$
L_1^c	n_1	n_1	n_1
$L_1 L_2$	$poly(n_1, n_2)$	$n_1 n_2$	$(2n_1 - 1)2^{n_2 - 1}$
L_1^*	$poly(n_1)$	$(n_1 - 1)^2 + 1$	$2^{n_1 - 1} + 2^{n_1 - 2}$
$L_1 \sqcup\!\sqcup L_2$	$poly(n_1, n_2)$	$n_1 n_2$	$2^{n_1 n_2} - 1$
L_1^R	n_1	n_1	2^{n_1}
$P_{\Sigma_0}(L_1)$	$e^{O(\sqrt{n_1 \cdot \ln n_1})}$	n_1	$3 \cdot 2^{n_1 - 2} - 1$

Table 7.3: State complexity of operations on regular languages under Parikh and standard equivalence.

7.2 Future works

The research described in this thesis can be extended along several directions as follows:

- As in the case of the intersection, the complement does not commute with the Parikh mapping. The noncommutativity in the case of complement is illustrated in the inequality $\psi((a^*b^*)^c) \neq (\psi(a^*b^*))^c$. The left-hand side is the linear set $\mathbb{N} \times \mathbb{N}$, while the right-hand side is the empty set. However, semilinear sets are closed under complement [GS64]. Hence the problem arises of stating how small can we make a 1dfa accepting a language whose Parikh image is the complement of the Parikh image of the language accepted by a given 1dfa.

  The fact that the Parikh image of a language accepted by a 1nfa is semilinear and the closure property of semilinear sets under complement makes this problem meaningful.

- Another possible extension is the minimization problem. Given a deterministic automaton, finding the smallest Parikh equivalent deterministic automaton is an open question. More precisely, given a 1dfa A, we want to know if there is a 1dfa B such that:

  – B is Parikh equivalent to A.

  – the number of states of B is less than number of states of A.

  It could be interesting to find an efficient algorithm to solve it.

- It could also be interesting to study computational complexity aspects of Parikh equivalent conversions.

  As we know, in the case of Cnfgs with h variables all known translations from cfg to semilinear set yield at least exponentially many linear sets [Esp97, Koz97, SSMH04, VSS05]. All these constructions run in exponential time. Kopczyński and To in [KT10, Proposition 10] show that this cannot be improved even for cfgs over a fixed unary alphabet. More precisely, they show that the number of linear sets for Parikh images of cfgs could be exponential in the size of the cfgs. More details can be found in [To10a, Theorem 2.2.12, Proposition 7.3.9].

  Furthermore, we use the result proven by Esparza et al. [EGKL11], which gives the state cost of the conversion of Cnfgs into Parikh equivalent 1nfas (Theorem 5.1.3), but the computational complexity of their conversion is not known.

- An interesting decision problem is the following: given two automata, are they Parikh equivalent?

  This problem brings us to study the equality problem for semilinear sets, i.e., whether two semilinear set representations are equivalent. It is shown that the equivalence problem for semilinear sets (these sets are exactly the Presburger sets) is decidable in polynomial space [Huy80]. Hence we can solve it in deterministic time $2^{P(N)}$, where P is a fixed polynomial and N is the input size. More precisely, N is the sum of the sizes of representations of the semilinear sets being considered [GI79, Thm. 4.2].

  Furthermore, in the case of 1nfas with n states and fixed alphabet size, Kopczyński and To in [KT10, Thm. 8] gave a polynomial-time algorithm for computing such Parikh images. Hence the normal form for the Parikh image of 1nfas presented in Lemma 4.0.2 can be also obtained in polynomial time. Observe that the Parikh image of 1nfas in Lemma 4.0.2 is the union of a polynomial number of linear sets. We can conclude that the problem whether two automata are Parikh equivalent, is decidable in deterministic time $2^{p(n)}$, where p is a fixed polynomial and n is the input size.

- What we can say concerning the Parikh equivalence of two cfgs?

  All translations known to us from cfg to semilinear set run in an exponential time and may output a union of exponentially many linear sets in the worst case. Because of the exponential number of linear sets we obtain an exponential size of the semilinear set. So, the problem whether two cfgs are Parikh equivalent, is decidable in deterministic time $2^{2^{q(h)}}$, where q is a polynomial function on h (the size of the grammar). Hence, we can conclude according to these considerations that, the computational time to decide the Parikh equivalence of two cfgs is double exponential.

- Finally, in this thesis we have treated the conversion from 1nfa into Parikh equivalent 2dfa, but what happens in the other direction? More precisely, given a two-way deterministic automaton, how many states are necessary and sufficient to build a Parikh equivalent one-way deterministic or nondeteministic automaton?

Bibliography

[AÉI02] L. Aceto, Z. Ésik, and A. Ingólfsdóttir. A Fully Equational Proof of Parikh's Theorem. *RAIRO - Theoretical Informatics and Applications*, 36(2):129–153, 2002. [cited at p. 1]

[Ber80] P. Berman. A note on sweeping automata. In Jaco de Bakker and Jan van Leeuwen, editors, *Automata, Languages and Programming*, volume 85 of *Lecture Notes in Computer Science*, pages 91–97. Springer, 1980. [cited at p. 25]

[BH96] A. Bouajjani and P. Habermehl. Constrained Properties, Semilinear Systems, and Petri Nets. In *CONCUR'96: Concurrency Theory*, volume 1119 of *Lecture Notes in Computer Science*, pages 481–497. Springer, 1996. [cited at p. 5]

[BL77] P. Berman and A. Lingas. On complexity of regular languages in terms of finite automata. Technical Report 304, Instytut Podstaw Informatyki Warszawa, 1977. [cited at p. 25]

[Cho56] N. Chomsky. Three Models for the Description of Language. *IRE Transactions on Information Theory*, 2(3):113–124, 1956. [cited at p. 11, 13]

[Cho59] N. Chomsky. On Certain Formal Properties of Grammars. *Information and Control*, 2(2):137–167, 1959. [cited at p. 13]

[Chr86] M. Chrobak. Finite Automata and Unary Languages. *Theoretical Computer Science*, 47:149–158, 1986. Corrigendum, ibid. 302 (2003) 497–498. [cited at p. 1, 2, 25, 26, 39, 53, 54, 67]

[CSY02] C. Câmpeanu, K. Salomaa, and S. Yu. Tight Lower Bound for the State Complexity of Shuffle of Regular Languages. *Journal of Automata, Languages and Combinatorics*, 7(3):303–310, 2002. [cited at p. 13, 34, 60]

[Dan01] Z. Dang. Binary Reachability Analysis of Pushdown Timed Automata with Dense Clocks. In Gérard Berry, Hubert Comon, and Alain Finkel, editors, *Computer Aided Verification*, volume 2102 of *Lecture Notes in Computer Science*, pages 506–517. Springer, 2001. [cited at p. 5]

[EGKL11] J. Esparza, P. Ganty, S. Kiefer, and M. Luttenberger. Parikh's Theorem: A Simple and Direct Automaton Construction. *Information Processing Letters*, 111(12):614–619, 2011. [cited at p. 2, 3, 50, 51, 70]

[Esp97] J. Esparza. Petri Nets, Commutative Context-Free Grammars, and Basic Parallel Processes. *Fundamenta Informaticae*, 31(1):13–25, 1997. [cited at p. 2, 5, 70]

[Gaw11] P. Gawrychowski. Chrobak Normal Form Revisited, with Applications. In Béatrice Bouchou-Markhoff, Pascal Caron, Jean-Marc Champarnaud, and Denis Maurel, editors, *Implementation and Application of Automata*, volume 6807 of *Lecture Notes in Computer Science*, pages 142–153. Springer, 2011. [cited at p. 54]

[Gef07] V. Geffert. Magic numbers in the state hierarchy of finite automata. *Information and Computation*, 205(11):1652–1670, 2007. [cited at p. 39, 54]

[GI79] E. M. Gurari and O. H. Ibarra. The Complexity of the Equivalence Problem for Counter Machines, Semilinear Sets, and Simple Programs. In Michael J. Fischer, Richard A. DeMillo, Nancy A. Lynch, Walter A. Burkhard, and Alfred V. Aho, editors, *Proceedings of the Eleventh Annual ACM Symposium on Theory of Computing*, STOC '79, pages 142–152. Association for Computing Machinery, 1979. [cited at p. 70]

[GKK$^+$02] J. Goldstine, M. Kappes, C.M. Kintala, H. Leung, A. Malcher, and D. Wotschke. Descriptional Complexity of Machines with Limited Resources. *Journal of Universal Computer Science*, 8(2):193–234, 2002. [cited at p. 4, 21]

[GMP03] V. Geffert, C. Mereghetti, and G. Pighizzini. Converting Two-Way Nondeterministic Unary Automata into Simpler Automata. *Theoretical Computer Science*, 295(1-3):189–203, 2003. [cited at p. 25, 26]

[GMP07] V. Geffert, C. Mereghetti, and G. Pighizzini. Complementing two-way finite automata. *Information and Computation*, 205(8):1173–1187, 2007. [cited at p. 27, 34]

[Gol77] J. Goldstine. A simplified proof of Parikh's theorem. *Discrete Mathematics*, 19(3):235–239, 1977. [cited at p. 1]

[GP11] V. Geffert and G. Pighizzini. Two-way unary automata versus logarithmic space. *Information and Computation*, 209(7):1016–1025, 2011. [cited at p. 27]

[GR62] S. Ginsburg and H. G. Rice. Two Families of Languages Related to ALGOL. *Journal of the ACM*, 9(3):350–371, 1962. [cited at p. 39]

[Gru71] J. Gruska. Complexity and Unambiguity of Context-Free Grammars and Languages. *Information and Control*, 18(5):502–519, 1971. [cited at p. 23]

[Gru73] J. Gruska. Descriptional Complexity of Context-Free Languages. In *Mathematical Foundations of Computer Science: Proceedings of Symposium and Summer School, Strbské Pleso, High Tatras, Czechoslovakia, September 3-8, 1973.*, pages 71–83. Mathematical Institute of Slovak Academy of Science, 1973. [cited at p. 23]

[Gru76] J. Gruska. Descriptional complexity (of languages) a short survey. In Antoni Mazurkiewicz, editor, *Mathematical Foundations of Computer Science 1976*, volume 45 of *Lecture Notes in Computer Science*, pages 65–80. Springer, 1976. [cited at p. 23]

[GS64] S. Ginsburg and E. H. Spanier. Bounded Algol-Like Languages. *Transactions of the American Mathematical Society*, 113(2):333–368, 1964. [cited at p. 3, 4, 38, 62, 64, 65, 68, 69]

[GS66] S. Ginsburg and E. H. Spanier. Semigroups, Presburger formulas, and languages. *Pacific Journal of Mathematics*, 16(2):285–296, 1966. [cited at p. 2, 4]

[Har78] M. A. Harrison. *Introduction to Formal Language Theory.* Addison-Wesley, 1978. [cited at p. 12]

[HK10] M. Holzer and M. Kutrib. Descriptional Complexity - An Introductory Survey. In Carlos Martín-Vide, editor, *Scientific Applications of Language Methods*, chapter 1, pages 1–58. Imperial College Press, 2010. [cited at p. 21, 23]

[HMU01] J. E. Hopcroft, R. Motwani, and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, 2001. [cited at p. 13, 29]

[HS03] J. Hromkovic and G. Schnitger. Nondeterminism versus Determinism for Two-Way Finite Automata: Generalizations of Sipser's Separation. In Jos C. M. Baeten, Jan Karel Lenstra, Joachim Parrow, and Gerhard J. Woeginger, editors, *Automata, Languages and Programming*, volume 2719 of *Lecture Notes in Computer Science*, pages 439–451. Springer, 2003. [cited at p. 25]

[HU79] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley, 1979. [cited at p. 7]

[Huy80] D. T. Huynh. The complexity of semilinear sets. In Jaco de Bakker and Jan van Leeuwen, editors, *Automata, Languages and Programming*, volume 85 of *Lecture Notes in Computer Science*, pages 324–337. Springer, 1980. [cited at p. 1, 62, 63, 70]

[IBS00] O. H. Ibarra, T. Bultan, and J. Su. Reachability Analysis for Some Models of Infinite-State Transition Systems. In Catuscia Palamidessi, editor, *CONCUR 2000: Concurrency Theory*, volume 1877 of *Lecture Notes in Computer Science*, pages 183–198. Springer, 2000. [cited at p. 5]

[JM12] G. Jirásková and T. Masopust. On a structural property in the state complexity of projected regular languages. *Theoretical Computer Science*, 449(31):93–105, 2012. [cited at p. 13, 34, 61]

[Kap05] C. Kapoutsis. Removing Bidirectionality from Nondeterministic Finite Automata. In Joanna Jędrzejowicz and Andrzej Szepietowski, editors, *Mathematical Foundations of Computer Science 2005*, volume 3618 of *Lecture Notes in Computer Science*, pages 544–555. Springer, 2005. [cited at p. 25]

[Kle56] S. C. Kleene. Representation of Events in Nerve Nets and Finite Automata. In Claude Shannon and John McCarthy, editors, *Automata Studies*, pages 3–41. Princeton University Press, Princeton, NJ, 1956. [cited at p. 19]

[KMP14] M. Kutrib, A. Malcher, and G. Pighizzini. Oblivious Two-Way Finite Automata: Decidability and Complexity. *Information and Computation*, 237:294–302, 2014. [cited at p. 17]

[Koz97] D. Kozen. *Automata and Computability.* Undergraduate Texts in Computer Science. Springer New York, 1997. [cited at p. 70]

[KP12] C. A. Kapoutsis and G. Pighizzini. Reversal Hierarchies for Small 2DFAs. In Branislav Rovan, Vladimiro Sassone, and Peter Widmayer, editors, *Mathematical Foundations of Computer Science 2012*, volume 7464 of *Lecture Notes in Computer Science*, pages 554–565. Springer, 2012. [cited at p. 17]

[KT10] E. Kopczyński and A. W. To. Parikh Images of Grammars: Complexity and Applications. In *Logic in Computer Science (LICS), 2010 25th Annual IEEE Symposium on*, pages 80–89. IEEE Computer Society, 2010. [cited at p. 39, 40, 70]

[Lot02] M. Lothaire. *Algebraic Combinatorics on Words*, volume 90 of *Encyclopedia of Mathematics and its Applications*. Cambridge University Press, 2002. [cited at p. 10]

[LP12] G. J. Lavado and G. Pighizzini. Parikh's Theorem and Descriptional Complexity. In Mária Bieliková, Gerhard Friedrich, Georg Gottlob, Stefan Katzenbeisser, and György Turán, editors, *SOFSEM 2012: Theory and Practice of Computer Science*, volume 7147 of *Lecture Notes in Computer Science*, pages 361–372. Springer, 2012. [cited at p. 2, 53]

[LPS12] G. J. Lavado, G. Pighizzini, and S. Seki. Converting Nondeterministic Automata and Context-Free Grammars into Parikh Equivalent Deterministic Automata. In Hsu-Chun Yen and Oscar H. Ibarra, editors, *Developments in Language Theory*, volume 7410 of *Lecture Notes in Computer Science*, pages 284–295. Springer, 2012. [cited at p. 4]

[LPS13] G. J. Lavado, G. Pighizzini, and S. Seki. Converting Nondeterministic Automata and Context-free Grammars into Parikh Equivalent One-way and Two-way Deterministic Automata. *Information and Computation*, 228-229:1–15, 2013. [cited at p. 4]

[LPS14] G. J. Lavado, G. Pighizzini, and S. Seki. Operational State Complexity under Parikh Equivalence. In Helmut Jürgensen, Juhani Karhumäki, and Alexander Okhotin, editors, *Descriptional Complexity of Formal Systems*, volume 8614 of *Lecture Notes in Computer Science*, pages 294–305. Springer, 2014. [cited at p. 4]

[Lup63] O.B. Lupanov. A comparison of two types of finite automata. *Problemy Kibernet*, 9:321–326, 1963. (in Russian). German translation: Über den Vergleich zweier Typen endlicher Quellen, Probleme der Kybernetik 6, 329–335 (1966). [cited at p. 1, 22, 24]

[MF71] A. R. Meyer and M. J. Fischer. Economy of Description by Automata, Grammars, and Formal Systems. In *Proceedings of the 12th Annual Symposium on Switching and Automata Theory (Swat 1971)*, SWAT '71, pages 188–191. IEEE Computer Society, 1971. [cited at p. 1, 22, 24]

[Mic81] S Micali. Two-Way Deterministic Finite Automata are Exponentially More Succinct Than Sweeping Automata. *Information Processing Letters*, 12(2):103–105, 1981. [cited at p. 25]

[Moo71] F.R. Moore. On the bounds for state-set size in the proofs of equivalence between deterministic, nondeterministic, and two-way finite automata. *IEEE Transactions on Computers*, C-20(10):1211–1214, 1971. [cited at p. 1, 22, 24]

[MP01] C. Mereghetti and G. Pighizzini. Optimal Simulations between Unary Automata. *SIAM Journal on Computing*, 30(6):1976–1992, 2001. [cited at p. 25, 26]

[Par66] R. J. Parikh. On Context-Free Languages. *Journal of the ACM*, 13(4):570–581, 1966. [cited at p. 1, 2, 4, 39]

[PJ91] M. Presburger and D. Jabcquette. On the completeness of a certain system of arithmetic of whole numbers in which addition occurs as the only operation. *History and Philosophy of Logic*, 12(2):225–233, 1991. [cited at p. 5]

[Pre29] M. Presburger. Über die Volständigkeit eines gewissen Systems der Arithmetik ganzer Zahlen, in welchem die Addition als einzige Operation hervortritt. *In Sparawozdanie z I Kongresu matematyków krajów slowianskich*, 395(2):92–101, 1929. [cited at p. 5]

[PS02] G. Pighizini and J. Shallit. Unary Language Operations, State Complexity and Jacobsthal's Function. *International Journal of Foundations of Computer Science*, 13(1):145–159, 2002. [cited at p. 3, 59, 60]

[PSW02] G. Pighizzini, J. Shallit, and M. Wang. Unary Context-free Grammars and Pushdown Automata, Descriptional Complexity and Auxiliary Space Lower Bounds. *Journal of Computer and System Sciences*, 65(2):393–414, 2002. [cited at p. 3, 39, 51, 53]

[RS59] M. Rabin and D. Scott. Finite automata and their decision problems. *IBM Journal of Research and Development*, 3(2):114–125, 1959. [cited at p. 1, 17, 22, 24, 25]

[Sha08] J. O. Shallit. *A Second Course in Formal Languages and Automata Theory.* Cambridge University Press, 2008. [cited at p. 7]

[She59] J. C. Shepherdson. The Reduction of Two-Way Automata to One-Way Automata. *IBM Journal of Research and Development*, 3(2):198–200, 1959. [cited at p. 24, 25]

[Sip80] M. Sipser. Lower Bounds on the Size of Sweeping Automata. *Journal of Computer and System Sciences*, 21(2):195–202, 1980. [cited at p. 25]

[SS78] W. J. Sakoda and M. Sipser. Nondeterminism and the Size of Two Way Finite Automata. In *Proceedings of the Tenth Annual ACM Symposium on Theory of Computing*, STOC '78, pages 275–286, New York, NY, USA, 1978. ACM. [cited at p. 17, 25]

[SSMH04] H. Seidl, T. Schwentick, A. Muscholl, and P. Habermehl. Counting in Trees for Free. In Josep Díaz, Juhani Karhumäki, Arto Lepistö, and Donald Sannella, editors, *Automata, Languages and Programming*, volume 3142 of *Lecture Notes in Computer Science*, pages 1136–1149. Springer, 2004. [cited at p. 70]

[To10a] A. W. To. *Model Checking Infinite-State Systems: Generic and Specific Approaches.* PhD thesis, School of Informatics, University of Edinburgh, August 2010. [cited at p. 2, 70]

[To10b] A. W. To. Parikh Images of Regular Languages: Complexity and Applications. arXiv:1002.1464v2, February 2010. [cited at p. 40]

[VSS05] K. Verma, H. Seidl, and T. Schwentick. On the Complexity of Equational Horn Clauses. In Robert Nieuwenhuis, editor, *Automated Deduction - CADE-20*, volume 3632 of *Lecture Notes in Computer Science*, pages 736–736. Springer, 2005. [cited at p. 2, 70]

[Yu00] S. Yu. State complexity of regular languages. *Journal of Automata, Languages and Combinatorics*, 6:221–234, 2000. [cited at p. 3, 34, 58, 60]

[YZS94] S. Yu, Q. Zhuang, and K. Salomaa. The State Complexities of Some Basic Operations on Regular Languages. *Theoretical Computer Science*, 125(2):315–328, 1994. [cited at p. 3, 34, 58, 60, 61]