N° d'ordre: 00000

THÈSE

présentée

devant l'Université de Rennes 1

pour obtenir

le grade de : Docteur de l'Université de Rennes 1
Mention Informatique

par

Antoine Meyer

Équipes d'accueil :	Groupe Galion – IRISA Équipe Modélisation et Vérification – LIAFA (Université de Paris 7)
École Doctorale :	Matisse
Composante universitaire :	IFSIC

Titre de la thèse :

Finitely Presented Infinite Graphs

À soutenir le 14 octobre 2005 devant la commission d'examen

MM. :	Kamal	Lodaya	Rapporteurs
	Denis	Lugiez	
MM. :	Jean-Claude	Raoult	Examinateurs
	Ahmed	Bouajjani	
	Didier	Caucal	

Contents

Introduction

The object of this thesis is to contribute to a systematic study of families of *finitely presented* infinite mathematical objects, namely *infinite graphs*. Special attention is devoted to the *structural properties* of each family, as well as their comparison to each other. Apart from its theoretical motivation, the underlying goal of this work is to help provide new infinite models for computer science objects like programs, towards a potential use in program verification.

Model Checking Checks Models

A very general formulation of the verification problem for computer programs or similar systems is: 'Does the behavior of program X respect property Y?' where Y may refer to the correctness of computations performed by X, to the fact that program X does not enter an infinite loop or a sterile blocking configuration, or to any kind of expected behavior of X. Note that we are mostly concerned with properties which do not refer to the language or implementation details of X but rather to its computations. In most cases, programmers rely on simple ways to answer this kind of questions: they can either trust their programming skills, run a few tests on a subset of possible inputs, or prove its correctness by hand.

In some cases however, one wishes (or needs) to perform a more systematic verification of the correctness of a program with respect to its specifications. To answer this need, several research fields devoted to the improvement of programs' correctness have emerged, either by automatic generation of relevant test cases [Mye79], semi-automatic theorem proving [NPW02, BC04, ORS92], proof-carrying code [Nec97], abstract interpretation [CC77] or various other static or dynamic formal program analyzes.

Another well-known approach, which is more relevant to the present work, is the so-called field of *automatic verification*, which attempts to prove properties of programs using algorithms, i.e. without any human intervention. In this framework, a program X is associated to a model[1] M, which is a logical structure

[1] In this sentence, the word 'model' refers to the mathematical formalization of a concrete object, here a program.

formally describing its behavior (or semantics). On the other hand, a property Y to be verified on X is formally expressed as a formula ϕ in a logical language interpreted over the signature of structure M. For instance, a model for a program could be the logical structure associated to a finite automaton with labeled states, and the logic used to express ϕ could be the temporal logic LTL. The problem of verifying program X with respect to property Y then translates into a *model checking* problem[2]:

$$\text{Program } X \text{ has property } Y \iff M \models \phi.$$

Of course, this logical equivalence has to be taken with some distance. For it to be valid, one has to ensure that M faithfully represents the behavior of program X, and that ϕ indeed expresses property Y, which is a modeling problem. Furthermore, assuming that the formalization of X and Y is correct, it is a well-known consequence of the work of Turing and Church [Tur36, Chu36] that for any general enough class of programs and properties, this problem is undecidable.

From this observation, there are two main ways to proceed. One is to consider restricted classes of programs and properties for which models with a decidable model checking problem are known, and the other is to *abstract* too general programs towards a class of models one is able to verify. In the latter case however, because of the abstraction, the truth value of a formula over a model need no longer match that of the corresponding property over the original program. Several techniques were developed to still be able to deduce properties of programs from their abstract models, for instance by proving that any property true of the model must be true of the program, or conversely (one speaks of *safe abstraction*).

Automata as Models for Programs

Well established and successful automatic verification techniques using finite automata theory as a general theoretical framework for the verification of programs have been proposed by several authors (for instance [CES86, VW86]). Finite tree or word automata enjoy nice decidability properties and a long history of study as one of the main tools in language theory and other connected areas (see for instance [Ber79, Sak03]). Most importantly, many algorithms with reasonable complexities were adapted or developed to solve their model checking problem in various cases. Finally, several standard techniques for the safe abstraction of sequential programs as finite automata were proposed.

However, a drawback of such finite-state verification techniques is that the class of programs they can model is quite restricted. Even used as abstractions

[2]In logic, a *model* of a given formula is a structure over which the formula holds. This meaning of the word model slightly differs from the previous one.

for programs, finite-state automata present a severe expressiveness limitation, since any source of infiniteness in programs has to be abstracted away: data variables over unbounded domains, temporization, concurrency, dynamic creation of new processes or other aspects of a program which account for an infinite number of configurations cannot be satisfyingly represented by finite automata. Consequently, one cannot hope using only finite automata to verify program properties which specifically rely on these infinite aspects.

To address this limitation, recent advances in the field of automatic verification are more and more concerned with designing new techniques over infinite objects. One of the main ideas to support these new techniques was to use more expressive formalisms, some borrowed from related theories, like pushdown automata, automata with counters or clocks, communicating automata, rewriting systems, etc. However, a difficulty inherent to such infinite-state formalisms is that one may not as easily visualize the structure of their computations as in the finite case. A good way to overcome this difficulty is to consider the family of *infinite graphs* such infinite-state formalisms generate.

Infinite Graphs

There are several ways one may define an infinite graph associated to the finite description of a given machine or automaton. In [MS85], Muller and Schupp considered the family of *pushdown graphs*, whose vertices represent the reachable configurations of a pushdown automaton, and whose edges illustrate the effect of the automaton's transitions. A key feature of their approach was to give an alternative characterization of this family of graphs, independently of the actual values of configurations: they proved that the set of all connected graphs one could obtain by successively removing 'layers' of a pushdown graph, starting from any vertex, is finite. This finite decomposition property allowed them, using the decidability of the monadic second-order logic (MSO) over the (infinite) complete binary tree proved by Rabin [Rab69], to deduce that pushdown graphs have a decidable MSO theory, or in other words a decidable model checking problem for this logic.

In [Cou90], Courcelle showed the same result for the strictly more general family of *HR-equational* graphs, which are the graphs generated by deterministic graph grammars, again using Rabin's theorem. In some of his works [Cau92, Cau96], Caucal described two alternative characterizations of pushdown graphs and HR-equational ones, using either rewriting systems or graph transformations from the complete binary tree preserving the decidability of the monadic second-order theory. This latter characterization allowed him to provide a new proof of the decidability of MSO over these families, as well as an extension of these result to the strictly more general family of *prefix-recognizable* graphs. In the following

years, several other interesting families of graphs were defined and studied using various characterizations, for instance *automatic* [BG00] and *rational* graphs [Mor00], higher-order prefix-recognizable graphs [CK02a], or various families of term rewriting graphs.

These results shed some light on the advantages of considering infinite graphs *up to isomorphism*, i.e. without enforcing a particular naming for the vertices of a graph, only to retain and study its *structure*. There are several simple explanations for this. The simplest one is that one would not wish to differentiate two graphs whose only difference lies in the choice of an alphabet to code its vertices, if the vertex domain is a set of words or terms, or in the choice of a number base if vertices represent integers.

In a more concrete setting, consider two graphs modeling the behavior of two reactive programs receiving inputs from a user or an environment. Vertices of each graph represent the internal state of the corresponding program, including variables, execution stacks and program counter, as it waits for an external input; and edges represent the effect of a given input on the program. If we are interested in determining whether these two programs are equivalent from a behavioral point of view, there is no point in comparing their actual implementation, only the structure of their computations.

From a theoretical point of view, a direct consequence of considering families of graphs up to isomorphism is that the same family may have several different sets of representatives. Several natural questions arise from this. One is to determine whether certain formalisms define the same family of graphs up to isomorphism, or to compare families of graph representatives to each other. Another important objective is to deduce properties of a family of graphs from the properties of its different sets of representatives. Third, since infinite graphs can be so easily connected with other areas of theoretical computer science, one should hope that their study will bring new results and applications in connected fields, like for instance language or complexity theory, and verification.

Outline of the Thesis

Introduction to the Field

After presenting a few necessary notions concerning languages, automata, relations and logic in Chapter 1, we give in Chapter 2 a brief overview of the current extent of knowledge in the field of infinite graphs theory. This chapter first exposes several ways in which finite formalisms can be used to describe the structure of infinite graphs, either internally by explicitly stating the edge relations of a graph, or in an external fashion by directly describing the *structure* of graphs, either by applying transformations to a simpler graph or using other types of

generation mechanisms. We also explain some of the specificities of each of these characterizations.

In a second time, we present a few of the most well-known and well-studied families of infinite graphs. We begin with the definition of a hierarchy of families whose main property is that their graphs have a decidable monadic second-order theory. This includes Muller and Schupp's pushdown graphs [MS85], Courcelle's HR- and VR-equational graphs [Cou90] and Caucal's prefix-recognizable hierarchy [Cau96, CK02a]. Most of the decidability results over these families make use of early theorems by Büchi [Büc62] and Rabin [Rab69] over the infinite string and the complete binary tree respectively. Then, we introduce the family of rational graphs first studied by Morvan [Mor00], and some of its sub-families (among which the word-automatic graphs [BG00]). Finally, we mention the graphs associated to Turing machines [Cau03], and a possible extension of some of these families to graphs whose vertices are represented by terms (for instance the ground term rewriting graphs [Löd02] or the term-automatic graphs [BG00]).

We conclude with a few words concerning current lines of research in the field of verification of infinite-state systems, and their links with the theory of infinite graphs.

Term-Rewriting Systems with a Rational Derivation

Chapter 3 is devoted to the characterization of families of term rewriting systems whose derivation relation (i.e. the relation consisting of all pairs of terms whose first component can be rewritten in any number of steps into the second) is recursive and can be finitely represented. We extend a classification of word rewriting systems based on possible overlappings of rewriting rules proposed in [Cau00] to the case of terms. We show that, for an appropriate notion of rationality for relations over tuples of terms, this study allows to characterize three families of term rewriting systems whose derivation relations are rational. Among the consequences of our results, we also obtain effective regularity preservation properties for these systems.

Numerous works deal with term rewriting systems. Among the closest to our approach, we can mention for instance [GV98] and [TKS00], which more specifically investigate the regularity preservation properties of systems. On the contrary, Dauchet and Tison extensively studied the derivations of ground term rewriting systems, i.e. systems whose rules do not contain variables [DT85]. In particular, they proved that ground systems have a decidable first order theory with reachability [DHLT90] by explicitly building their derivation relation. From another point of view, [Löd02] and [Col02] investigated the properties of transition graphs of ground term-rewriting systems and compared this family of graphs with respect to other well-known families.

A possible application of our results concerns the theme of symbolic model checking, whose main idea is to represent regular sets of configurations by finite word automata and system transitions by rewrite rules or transducers (see for example [BJNT00]). This field is currently being extended to systems with richer topologies, like trees [AJMD02, BT02]. A central problem relevant to this method is to compute the set of configurations reachable in any number of steps when starting from a *regular* set of configurations (for instance a regular term language).

The work presented in this chapter was presented at the 2004 FoSSaCS conference in Barcelona and published in its proceedings [Mey04]. A long version of this paper was recently submitted for journal publication, and is currently being reviewed.

Infinite Automata for Context-Sensitive Languages

Chapter 4 is an in-depth study of two families of infinite graphs seen as 'real-time' automata for context-sensitive languages, in the sense that they do not contain ε-labeled edges.

First, we investigate the family of rational graphs and some of its sub-families. We propose a new proof of the result by [MS01, Ris02][3] stating that the languages of rational graphs are the context-sensitive languages, which has the advantage to be self-contained (contrary to the original proof). We also consider meaningful structural restrictions on the degree and number of initial vertices of such graphs when they are considered as language acceptors.

In a second time, we introduce the family of linearly bounded graphs as the transition graphs of linearly bounded machines. As such, it is easy to see that the traces of these graphs are exactly the context-sensitive languages. We provide alternative characterizations of this family, and compare it with similar existing work [KNU02, Cau03].

For both families, we also consider the case of deterministic context-sensitive languages and characterize subsets of these families of graphs which accept precisely these languages. An interesting fact about deterministic context-sensitive languages is that their equivalence with non-deterministic languages is unknown [Kur64]. We conclude this chapter by comparing both families of graphs: it turns out that rational graphs and linearly bounded graphs are incomparable in general, but that even though they are of a very different nature, all bounded-degree rational graphs are bounded-degree linearly bounded graphs, and this inclusion is strict.

The material presented in this chapter is the product of joint work with Arnaud Carayol. All these results were submitted for journal publication [CM05a,

[3] A revised version of these results was recently published in [MR05].

CM05c] and are currently being reviewed. A shorter version of [CM05c] was presented at the 2005 MFCS conference (Gdansk), and published in the conference's proceedings [CM05b].

Reachability Analysis of Higher-Order Context-Free Processes

Finally, we present in Chapter 5 a direct symbolic reachability analysis algorithm for a subfamily of higher-order pushdown systems. This work's motivation concerns the computation of all ancestors of a given set of configurations encoded as a regular set of words. We show that techniques used in [BEM97] for the reachability analysis of pushdown systems can be extended to the so-called class of higher-order context-free processes, which are the natural higher-order extension of ordinary context-free processes (also called basic process algebras). These classes are obtained when considering (higher-order) pushdown systems with a single control state.

As an application of these results, we provide a model checking technique over the configuration graphs of higher-order context-free processes for the existential fragment of the CTL temporal logic restricted to the EU and EX modalities.

This work was done in common with Ahmed Bouajjani. It was presented at the 2004 FSTTCS conference in Chennai, and published in its proceedings [BM04].

Chapter 1

Preliminary Notions

In this chapter, we recall some of the fundamental concepts and notations which are used throughout this document. After giving the definitions of words, terms and graphs, we present some basic facts about rewriting systems over words and terms, and a brief overview of the well-known Chomsky hierarchy of languages. In particular, we recall a strict hierarchy of four families of acceptors for each of these families of languages. We then present the definitions of several classes of binary relations over words and terms, characterized either by 'algebraic' definitions or by automata accepting them. We end this chapter with an exposition of the first-order logic and monadic second order logic over graphs and some of their variations, as well as a brief summary of some of the concepts used in automata-theoretic verification.

1.1 Bestiary

1.1.1 Words

We consider finite sets of symbols, or *letters*, called *alphabets*. Finite sequences of letters are called *words*, and sets of words *languages*. In the following, whenever possible, we will use capital Greek letters Σ and Γ to denote finite alphabets, uppercase or lowercase Latin characters a, b, $c\ldots$ to denote symbols in these alphabets, and lowercase u, v and w to denote words.

A word u over alphabet Σ can be seen as a tuple $(a_1, \ldots, a_n)$ of elements of Σ, or equivalently as a mapping between some interval $[1, n]$ of integers and the set Σ. It is usually written $a_1 \ldots a_n$. Its i-th letter is denoted by $u(i) = a_i$. The set of all words over Σ is written Σ^*. The number of letters occurring in u is its length, written $|u|$ (here $|u| = n$). The unique word of length 0, called the *empty word*, is written ε. The concatenation of two words $u = a_1 \ldots a_n$ and $v = b_1 \ldots b_m$ is the word $uv = a_1 \ldots a_n b_1 \ldots b_m$. A word u is a *prefix* of another word v is there

exists some w such that $uw = v$. In that case, w is called a *suffix* of v. Any prefix of w (or suffix of u) is called a *factor* of v.

The concatenation operation extends to sets of words: for all $A, B \subseteq \Sigma^*$, AB stands for the set $\{uv \mid u \in A \text{ and } v \in B\}$. By a slight abuse of notation, we will simply denote by u the singleton $\{u\}$. Similarly, Σ can also be used to denote the set of 1-letter words, and more generally Σ^n the set of n-letter words.

Monoids. The algebraic structure underlying words is the *monoid*. A monoid M is simply a set, potentially infinite[1], together with an associative internal product operation with a neutral element $1_M \in M$. A subset S of a monoid M is a set of *free generators* for M if all elements of M admit exactly one decomposition as a product of elements of S. If furthermore S is finite, M is said to be finitely generated by S.

Example 1.1. The set $\mathbb{N}$ of positive integers is a free monoid for the addition operation. Its neutral element is 0. It is free and finitely generated by the singleton $\{1\}$.

Example 1.2. The set Σ^* of all words over an alphabet Σ is a monoid for concatenation. It is free, and finitely generated by Σ, and its neutral element is the empty word ε. In language theory, the characterization of words as elements of the free monoid is useful to define families of languages and relations (cf. Section 1.4).

1.1.2 Terms

Terms are more general mathematical objects than words, which intuitively represent sequences of applications of functions with several arguments. Terms are defined using *ranked alphabets*, i.e. alphabets whose symbols have *arities*. The arity of a symbol f is $a(f)$. If $a(f) = n$, we sometimes denote f by $f^{(n)}$. Symbols of arity 0 are called *constants*.

Let $F = \bigcup_{n \geq 0} F_n$ be a finite ranked alphabet, each F_n being the set of symbols of F of arity n, and X be a finite set of *variables* disjoint from F (all sets F_n are obviously disjoint). Variables are considered of arity 0. The set of finite first-order terms on F with variables in X, written $T(F, X)$, is the smallest set including X such that $f \in F_n \ \wedge \ t_1, \ldots, t_n \in T(F, X)$ implies $ft_1 \ldots t_n \in T(F, X)$. Words can be seen as terms over a ranked alphabet whose symbols have arity exactly 1 and whose last symbol is a variable. The *height* of a term $ft_1 \ldots t_n \in T(F, X)$ is inductively defined as $1 + max(h_1, \ldots, h_n)$, where each h_i is the height of sub-term t_i. To make the reading easier, $ft_1 \ldots t_n$ will sometimes be written $f(t_1, \ldots t_n)$.

[1] But at most countable in the scope of this work.

A term is called *proper* or *non-trivial* if it contains at least one symbol in F (i.e. it is not reduced to a variable). Terms containing no variable are called *ground*. The set of ground terms is noted $T(F, \emptyset)$ or simply $T(F)$. The set of variables actually occurring in a term t is $V(t)$. A term t is *linear* if each of its variables occurs only once. A linear term t with n variables is called a *n-context*, its variables are sometimes noted $\Box_1, \ldots, \Box_n$, read from left to right in t. The set of n-contexts is denoted by $C_n(F)$, the set of all contexts by $C(F)$.

A common operation on terms is *substitution*. A substitution is fully defined by a mapping σ from X to $T(F, X)$, and extended to terms as follows: we note $t\sigma$ the application of a substitution σ to a term t, which is done by replacing every occurrence of each variable x occurring in t by the term $\sigma(x)$. The set of substitutions over F and X is noted $S(F, X)$. We use the special notation $c[t]$ for the substitution of the unique variable of c by term t, where c is a 1-context. We say c is a prefix of $c[t]$, and t a suffix of $c[t]$, by analogy with words. If there is a substitution σ such that $t = t'\sigma$, we say t' is a *factor* of $c[t'\sigma]$.

Positions. Let $\mathbb{N}$ be the set of strictly positive integers, we call *position* any word in the set $\mathbb{N}^*$. The same way words can be seen as mappings from indexes to symbols, every term t in $T(F, X)$ can be represented as a mapping from a prefix-closed set of positions $Pos(t)$, called the *domain* of the term, to the set $F \cup X$. Let $t = ft_1 \ldots t_n$ be a term, ε denotes the position of the leftmost functional symbol f of t, and for all $k \in [1, n]$, $p \in \mathbb{N}^*$, position kp denotes the symbol occurrence at position p in t_k. We write $t(p) = f$ if the symbol occurring at position p in t is f. We denote by $pos(x, t)$ the set of positions at which the variable $x \in X$ occurs in term $t \in T(F, X)$.

In the following, we will use the prefix partial order on positions, noted $\geqslant$: let p and q be two positions, $p \geqslant q$ if there is some $q' \in \mathbb{N}^*$ such that $p = qq'$. If furthermore $q' \neq \varepsilon$, we write $p > q$.

Term words. By analogy with words, we call sequences of terms *term words*. The set of term words over F and X is written $T^*(F, X)$. All previous definitions (ground terms, linearity, contexts, substitutions and so on) naturally extend to term words. Additionally, for any term word $s = s_1 \ldots s_n$ and n-context t, we use $t[s]$ as a shorthand notation for the term word $t\sigma$, where σ is defined as the mapping $\{\Box_i \mapsto s_i \mid i \in [1, n]\}$, where $\Box_i$ denotes the i-th variable from the left in t. Notations are extended to sets of term words in the usual way.

Example 1.3. Let $F = \{f^{(2)}, g^{(1)}, a^{(0)}, b^{(0)}\}$. We have $F_2 = \{f\}, F_1 = \{g\}$ and $F_0 = \{a, b\}$. Let $t = ffab\ ga$ be a ground term in $T(F)$, also written $f(f(a, b), g(a))$. Let x, y be variables, term $c_1 = fxy$ is a 2-context and $c_2 = fxga$ a 1-context; both belong to $T(F, \{x, y\})$ and are prefixes of t. We can write $t = c_1\sigma$

and $t = c_2\sigma$ with $\sigma = \{x \mapsto fab, y \mapsto ga\}$, or shortly $t = c_1[fabga] = c_2[fab]$. Term word $fabga$ is a suffix of t.

1.1.3 Graphs

A labeled, directed and simple *graph* is a set $G \subseteq V \times \Sigma \times V$ where Σ is a finite set of labels and V an arbitrary countable set. An element (s, a, t) of G is an *edge* of *source* s, *target* t and *label* a, and is written $s \xrightarrow[G]{a} t$ or simply $s \xrightarrow{a} t$ if G is understood. An edge with the same source and target is called a *loop*. The set of all sources and targets of a graph form its *support* V_G, its elements are called *vertices*. This vision of graphs as sets of edges allows us to define the union, intersection and inclusion over graphs as the corresponding set operations. A graph included in another graph G is called a *subgraph* of G. The subgraph of G *induced* by a set of vertices $V' \subseteq V$ is simply the set of edges of G whose source and target both belong to V' (i.e. $G \cap (V' \times \Sigma \times V')$). One also speaks of the *restriction* of G to V'.

A sequence of edges $(s_1 \xrightarrow{a_1} t_1, \ldots, s_k \xrightarrow{a_k} t_k)$ with $\forall i \in [2, k],\ s_i = t_{i-1}$ is called a *path*. It is written $s_1 \xrightarrow{u} t_k$, where $u = a_1 \ldots a_k$ is the corresponding *path label*. Vertex s_1 is called the origin of the path, t_k its destination. A path is called a *cycle* if its origin and destination are the same vertex.

A graph is *deterministic* (resp. *co-deterministic*) if it contains no pair of edges having the same source (resp. target) and label. Given a vertex v, the number of edges whose source (resp. target) is v is its out-degree $d^+(v)$ (resp. in-degree $d^-(v)$). The sum of $d^+(v)$ and $d^-(v)$ is the degree $d(v)$ of v. The degree (resp. in-degree, out-degree) of a graph is the largest degree (resp. in-degree, out-degree) over all vertices of the graph's support, if it exists, otherwise we say it is infinite.

An equivalent way to define a graph G is directly as a *labeled transition system*, i.e. as a finite set of binary edge relations $\xrightarrow{a}$ over V, one for each label a in Σ. The graph G^{-1} defined as the family $(\xrightarrow{a}{}^{-1})_{a\in\Sigma}$ is the *inverse graph* of G. We write $\longrightarrow$ the adjacency relation defined as $\bigcup_{a\in\Sigma} \xrightarrow{a}$, $\longleftrightarrow$ its closure under symmetry. A subgraph of G whose support is closed under $\xleftrightarrow{*}$ is called a *connected component* of G. It is *strongly connected* if it is closed under $\xrightarrow{*}$. Intuitively, connected components are parts of a graph in which any vertex is reachable from any other by an undirected path (i.e. a path using edges and reversed edges). A vertex of a graph is called a *root* if there is a path from this vertex to any other vertex in the graph. A graph which has at least one root is said to be *rooted*.

Graphs and languages. One can associate a language to a graph by considering the set of all words labeling a path between two given sets of vertices.

Formally, we call *set of traces* of a Σ-labeled graph G between I and F, or simply *language* of G, the language $L(G, I, F) = \{u \in \Sigma^* \mid \exists s \in I, t \in F,\ s \xrightarrow[G]{u} t\}$.

An important question when considering the sets of traces of a graph or a family of graphs is to specify what kind of sets of initial and final vertices is allowed. It makes a great difference to consider only finite sets of vertices rather than infinite ones. We say two families of graphs F_1 and F_2 over the same vertex domain V are *trace-equivalent* with respect to families K_I and K_F of subsets of V if, for all $G \in F_i$, $I \in K_I$, $F \in K_F$, there exist a graph $H \in F_{3-i}$, $I' \in K_I$ and $F' \in K_F$ such that $L(G, I, F) = L(H, I', F')$.

The study of traces of families of infinite graphs is the key motivation of several works in the field of infinite graphs. We will discuss this point further in the next chapter. In Chapter 4, we will specifically investigate families of graphs whose traces are the context-sensitive languages (cf. Section 1.3.3).

Graph isomorphisms. One of the main aspects of this thesis is about comparing families of graphs, their structures, and some of their properties. There are several ways in which two graphs can be compared. Two graphs are obviously equal if they consist of the same set of edges, but one usually wants to consider less restrictive notions of equivalence.

The image of a graph G with support V_G by a mapping ϕ defined from V_G to some set V is the graph $\phi(G) = \{(\phi(s), a, \phi(t)) \mid (s, a, t) \in G\}$. Let H be a graph of support V, ϕ is a *graph morphism* from G to H if for all $(s, a, t) \in G$, $(\phi(s), a, \phi(t)) \in H$. If moreover ϕ is onto and ϕ^{-1} is also an onto morphism from H to G, then ϕ is called an *isomorphism* and we say G and H are isomorphic. The equivalence class $[G]$ of a graph G under isomorphism is the set of all graphs isomorphic to G. We say G is a *representative* of its equivalence class. We will often confuse G and its equivalence class, and write both simply as G.

Intuitively, two graphs are isomorphic if one can be obtained from the other by renaming each of its vertices with a new, unique name. In this thesis, unless otherwise stated, we will always consider families of graphs up to isomorphism. In other words, when comparing two families, we 'forget' the naming of vertices, only considering them as anonymous elements in a countable set, to retain only the structure of graphs. The next chapter will discuss this idea further.

Colored graphs. In some occasions, it will be convenient to associate extra information to some vertices in a graph, under the form of vertex labels, also called *colors*. Given a set C of colors, a C-colored graph is a pair (G, ϕ) where G is a graph and ϕ is a partial function mapping a color in C to some of the vertices of G. All previous definitions concerning graphs still hold in this extended case, simply by ignoring the colors of vertices. Colors are most often used for modeling

purposes, or when considering logics over graphs (cf. Section 1.5).

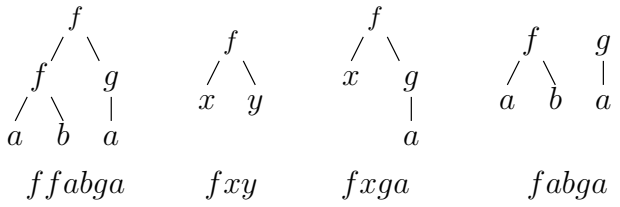
Terms and words seen as graphs. For the purpose of uniformity, we may see words and terms as particular instances of colored graphs. Terms in $T(F, X)$ are usually represented as finite ordered trees whose nodes are labeled by symbols in F or variables in X. Trees are rooted graphs in which there is exactly one path from the root to any other vertex in the graph. Let F be a ranked alphabet and t a term over F, one usually associates to t the colored tree

$$T = (\{(u, k, uk) \mid u, v \in Pos(t), k \in \mathbb{N}\}, \{u \mapsto f \mid t(u) = f\}).$$

The support of T is the domain of t. Each vertex of a T, called a *node*, is labeled with the functional symbol at the corresponding position in t, and its edges link a node representing a functional symbol to each of the nodes representing its arguments. For simplicity, we usually omit edge labels and orientation when drawing trees which represent terms, with the implicit convention that the successors of a node are ordered from left to right in ascending edge label order, and below the node itself (i.e. the root is on top, and leaves are at the bottom).

In the following, we will often consider a term and the tree which represents it as the same object, since both are completely equivalent. In particular, we will rather speak of term languages on one side and tree automata on the other, the latter accepting the former (cf. Section 1.3.5.2). Seeing terms as trees, term words can be seen as ordered forests. This should not lead to confusion.

Example 1.4. Terms t, c_1, c_2 and term word $fabgb$ from Ex. 1.3 can be represented by the following colored trees:



We already mentioned that words can be seen as terms over an alphabet whose symbols have arity one and whose last symbol is a variable. In this respect, they can also be seen as trees of out-degree bounded by one, or equivalently as connected graphs of degree 1.

1.2 Rewriting Systems

Rewriting systems are among the most general formalisms found in computer science to model word or term transformations[2] (see for instance [DJ90]). They generalize grammars, can represent the runs of finite automata, transducers, pushdown automata or even Turing machines, as we will see later on in this chapter. In fact, whenever it is possible, we will give an alternate definition of these heterogeneous formalisms as special cases of rewriting systems.

Word rewriting systems. A word rewriting system over alphabet Σ is a potentially infinite set of rewriting rules, i.e. pairs of words $(l, r) \in \Sigma^* \times \Sigma^*$. We denote by $Dom(R)$ (resp. $Ran(R)$) the set of left-hand sides (resp. right-hand sides) of R. A rule (l, r) *rewrites* a word w into w' by replacing an instance of its left-hand side l in w by r. More formally, the *rewriting* relation according to a system R is the relation

$$\underset{R}{\longrightarrow} := \{(ulv, urv) \mid (l, r) \in R \ \wedge \ u, v \in \Sigma^*\}.$$

The reflexive and transitive closure of this relation is called the *derivation* of R and written $\underset{R}{\overset{*}{\longrightarrow}}$. For all u, v with $u \underset{R}{\overset{*}{\longrightarrow}} v$, there exists a sequence of words $u_0 \dots u_n$, called a *derivation sequence*, such that

$$u = u_0 \underset{R}{\longrightarrow} u_1 \dots u_{n-1} \underset{R}{\longrightarrow} u_n = v.$$

A word w containing no left-hand side of any rule of a system R as a factor is called a *normal form* for R. A rewriting system for which all word w can be derived into a normal form is said to be *normalizing*, and *strongly normalizing* if this normal form is unique for any given w. The set of all normal forms of a system R is written $\mathrm{NF}(R)$.

Term rewriting systems. Term rewriting systems can be defined similarly as word rewriting systems: a term rewriting system R over some ranked alphabet F is composed of a potentially infinite set of rules (l, r), where l and r are terms in $T(F, X)$ such that $Var(r) \subseteq Var(l) \subseteq X$ (X is a set of variables supposed disjoint from F). We explicitly consider sets of rules which are closed under variable renamings in X, so that rules only differing by the names of their variables can be considered equal. The rewriting relation must be redefined using substitutions in order to take into account the fact that term rewriting rules may modify

[2] More general notions of rewriting adapted to graphs also exist, but they will not be presented here.

the structure of a term, swap branches, and so on. The rewriting relation of a rewriting system R is[3]

$$\underset{R}{\longrightarrow} := \{(c[l\sigma], c[r\sigma]) \in T(F) \times T(F) \mid (l,r) \in R \ \wedge \ c \in C_1(F) \ \wedge \ \sigma \in S(F,X)\}.$$

All previous definitions for word rewriting systems straightforwardly extend to terms. In case we want to specify that a rule (l,r) is used at some position p (resp. set of positions P), we use the notation $\underset{l,r}{\longrightarrow}^{p}$ (resp. $\underset{l,r}{\longrightarrow}^{P}$). A rewriting rule (l,r) is said to be *linear* if both l and r are linear terms. A system containing only linear rules is also called linear, and a system whose rules do not contain variables is called ground.

Labeled systems. In some occasions it will be useful to consider word or term rewriting systems whose rules carry labels in a finite alphabet Σ. In this new settings, rewriting rules are triples (l,a,r). Additionally to its rewriting relation $\underset{R}{\longrightarrow}$ defined as previously without taking the labels into account, a labeled system R defines a finite family of relations $(\underset{R}{\overset{a}{\longrightarrow}})_{a\in\Sigma}$. For any labeled system R, let R_a be the unlabeled system $R_a = \{(l,r) \mid (l,a,r) \in R\}$. We then simply define $\underset{R}{\overset{a}{\longrightarrow}}$ as $\underset{R_a}{\longrightarrow}$. These relations generate a monoid for the relational composition operation defined as $\underset{R}{\overset{uv}{\longrightarrow}} = \{(x,y) \mid \exists z, x \overset{u}{\longrightarrow} z \overset{v}{\longrightarrow} y\}$ with $u,v \in \Sigma^*$. The neutral element of this monoid is the identity relation $\underset{R}{\overset{\varepsilon}{\longrightarrow}}$ over the rewriting domain. A sequence $x_0 \overset{a_1}{\longrightarrow} x_1 \ldots \overset{a_n}{\longrightarrow} x_n$ is called a labeled derivation sequence, its label is the word $a_1 \ldots a_n$. We still write $\underset{R}{\overset{*}{\longrightarrow}}$ the unlabeled derivation of R. We often omit R in all these notations when it is clear from the context.

1.3 A Hierarchy of Languages and Acceptors

In this section, we give a few hints on one of the most widely used hierarchy in formal language theory, namely the Chomsky hierarchy of languages [Cho59]. Defined more than 40 years ago, it contains four of the most well-studied classes of languages, namely the *recursively enumerable*, *context-sensitive*, *context-free* and *regular* languages, which have numerous connections with other fundamental problems of computer science. In particular, all these families correspond to well-known classes of finite *acceptors*, sometimes called *automata* or *machines*, which characterize them. This section defines the four classes of grammars of

[3]Without loss of generality, we only consider rewriting of ground terms. Terms containing variables may be rewritten by considering variables as constants.

the Chomsky hierarchy defining these classes of languages, then details the main families of acceptors for each of them.

1.3.1 Chomsky's Hierarchy of Languages

The Chomsky hierarchy was initially defined using grammars, which are a special case of rewriting systems, in the aim to provide formal tools for the study of natural languages. A grammar is characterized by a tuple $G = (T, N, S, P)$, where N and T are two finite disjoint alphabets of non-terminal and terminal symbols respectively, $S \in N$ is an initial symbol called the *axiom*, and P is a finite word rewriting system over alphabet $(N \cup T)$ whose elements are usually called *grammar rules*. For simplicity, left-hand sides of rules of P are generally required to contain at least one non-terminal symbol.

We say a word $u \in T^*$ is *generated* by G if it can be derived by the rewriting system P from the axiom S. The *language* of the grammar is defined as the set of all terminal words which are generated by G:

$$L(G) = \{u \in T^* \mid S \xrightarrow[P]{*} u\}.$$

Remark 1.5. By definition of the rules of the grammar, a word containing only terminals can no longer be derived; it is a normal form for the rewriting system P. In fact, to explicitly distinguish terminal symbols from non-terminals is not an essential feature of grammars, which can be seen simply as rewriting systems.

The four classes in the Chomsky hierarchy are defined by putting additional constraints on the production rules of the grammars. Recall that a basic constraint is that all left-hand sides should contain at least one non-terminal symbol.

- A grammar is *unrestricted* or of *type 0* if no constraint is put on the form of its rules. Such grammars generate the *recursively enumerable* languages.
- A grammar is *context-sensitive* or of *type 1* if all its productions are of the form (v, w) with $|w| \geq |v|$. Its language is called *context-sensitive*, or of type 1.
- A grammar is *context-free* or of *type 2* if all its productions are of the form (A, w) with $A \in N$ and $w \in (N \cup T)^*$. Its language is called *context-free* or of type 2.
- A grammar is *left-linear* (or *right-linear*) if all its rules are of the form (A, Bu) or (A, u) (resp. (A, uB)), where A, B are non-terminal symbols and u is a terminal word. A grammar is called *regular* or of *type 3* if it is either left-linear or right-linear. The generated language is called a type 3 or *regular* language.

These four families of grammars generate a strict hierarchy of four families of languages, type 0 languages being the most general. We will see in the rest of this section that each of these families corresponds to a family of automata or machines accepting them. Turing machines accept all type 0 languages when their allowed tape space is not restricted, and type 1 languages when the memory they can use is at most linear in the size of the words they read. Type 2 languages are the languages accepted by pushdown automata, and type 3 languages by finite automata.

1.3.2 Turing Machines

Turing machines are a model proposed by Turing [Tur36] in the thirties to try and give a mathematical formalization of algorithms and automated computation. Turing machines intuitively consist of a finite control together with an auxiliary memory of unbounded size on which they can read and write symbols.

Turing machines are very expressive and accept the whole set of type 0 languages, called recursively enumerable languages. In fact, the famous thesis referred to as the 'Church-Turing thesis' is that Turing machines precisely capture our intuitive notion of algorithmic computation.

1.3.2.1 Definition

In this age of dynamic memory allocation, we give a somewhat different definition of Turing machines than the one most textbooks provide (for instance [HU79]).

In classical definitions, a Turing machine starts its computation with a finite input word written on an bi-infinite tape divided into cells, and proceeds by reading from and writing symbols to the tape. The variant of Turing machines we describe start with an empty tape with no cells, but are able to insert new cells or delete cells at any point during computation (in terms of data structures, think of the tape as a dynamic list rather than an array). Also, instead of starting their runs with an input word already written on their tape, they read input symbols one by one.

This notion is inspired by the classical notion of offline Turing machines, also called labeled Turing machines in [Cau03].

Definition 1.6 (Turing machine)**.** A Turing machine (in short TM) is a tuple $M = (\Sigma, \Gamma, [,], Q, q_0, F, \delta)$, where Σ and Γ are finite sets of *input* and *tape symbols* respectively, [and] $\notin \Gamma$ are *tape delimiters*, Q is a finite set of *control states*, $q_0 \in Q$ is the unique *initial state*, $F \subseteq Q$ is a set of *final states* and δ is a finite

set of *transition rules* of one of the forms:

$$p[\xrightarrow{l} q[+ \qquad pA \xrightarrow{l} qB\pm \qquad p] \xrightarrow{l} q]- \tag{1.1}$$

$$p[\xrightarrow{l} q[\qquad pA \xrightarrow{l} qB \qquad p] \xrightarrow{l} q] \tag{1.2}$$

$$pA \xrightarrow{l} q \qquad pA \xrightarrow{l} qBA \qquad p] \xrightarrow{l} qB] \tag{1.3}$$

with $p, q \in Q$, $A, B \in \Gamma$, $\pm \in \{+, -\}$ and $l \in \Sigma \cup \{\varepsilon\}$[4].

The tape of a Turing machine is composed of an arbitrarily large number of cells arranged in a line, each cell containing a symbol in Γ. At any point, the current *configuration* of the machine consists in its control state, the size and content of its tape, and the position of the read head along the tape. The cell currently faced by the head is referred to as the current cell in the following.

Each type of rules in the above definition describes a possible type of transition of the machine when either reading a symbol from the input ($l \in \Sigma$) or not ($l = \varepsilon$). A rule labeled by ε is called an ε-rule, it corresponds to an internal action of the machine. The left-hand side of a rule indicates what the control state and content of the current cell must be for the rule to be enabled. The right-hand side describes what action to perform when the rule is used.

A rule $pA \xrightarrow{l} qB\pm$ of type (1.1) rewrites the content of the current cell with B, steps into control state q and moves the head to the next cell (to the right if $\pm = +$, to the left if $\pm = -$) provided the current state is p and the current cell's content is A. Note that when A is a boundary symbol, we must have $B = A$ and the movement is restricted so that the head does not cross the boundary. Rule type (1.2) describes rewriting of the current cell content without any movement of the head. Again, boundaries are taken care of as special cases and may not be rewritten. Finally, rule type (1.3) describes the insertion of a new cell with content B to the left of the current cell, or the deletion of the current cell, provided the state is p and the current cell contains an A. If a new cell is added, it becomes the current cell, and if a cell is deleted its former right neighbor does. Think of cell deletion as deletion of an element from a list: even though the cell is 'deallocated', the list structure is preserved by gluing together the cells to the left and right.

1.3.2.2 Configurations and Runs

A very simple and useful way to represent Turing machine configurations is by words of the form uqv with $uv \in [\Gamma^*]$, $v \neq \varepsilon$ and $q \in Q$, where uv represents the tape content (enclosed in a pair of border symbols), q is the current control

[4]Note that this definition excludes rules erasing the boundaries or inserting cells outside the boundaries of the tape.

state and the head points to the tape cell containing the first letter of v. Let C_M denote the set of words of this form, which is the set of possible configurations of M. A convenient way to interpret Turing machine transition rules is as sets of rewriting rules over C_M. For all $l \in \Sigma \cup \{\varepsilon\}$, we define the labeled *transition relation* $\xrightarrow[M]{l}$ of M as the restriction to $C_M \times C_M$ of the rewriting relation of the labeled rewriting system

$$\begin{aligned}\{(pA, Bq) \mid pA \xrightarrow{l} qB+ \in \delta\} \ \cup \ \{(pA, qB) \mid pA \xrightarrow{l} qB \in \delta\} \\ \cup \{(CpA, qCB) \mid pA \xrightarrow{l} qB- \in \delta \wedge C \in \Gamma \cup \{[\}\} \\ \cup \{(pA, q) \mid pA \xrightarrow{l} q \in \delta\} \ \cup \ \{(pA, qBA) \mid pA \xrightarrow{l} qBA \in \delta\}.\end{aligned}$$

M starts each of its computations in its *initial configuration* c_0 represented by the word $[q_0]$. It corresponds to an empty tape, and a read head in control state q_0 pointing to the right boundary marker. An configuration is said to be *accepting*, or terminal, if its control state is in the set F. A sequence $c_0 \xrightarrow[M]{l_1} c_1 \dots c_{n-1} \xrightarrow[M]{l_n} c_n$ is called a *run* of M on input word $w = l_1 \dots l_n$. Notice that $|w| \leq n$ in general, since some of the l_i may be ε. A run is accepting if it ends in an accepting configuration. The *language* $L(M)$ of M is the set of all words w for which there exists an accepting run in M.

M is called *deterministic* if it has at most one run per input word. This feature is undecidable in general, but a sufficient condition is that no pair of different rules can be used from the same configuration, unless they are labeled by distinct letters, and both from the input alphabet (not by ε). It can be shown that any Turing machine can be transformed into a deterministic machine accepting the same language. A machine is *non-ambiguous* if it has at most one *accepting* run per input word.

Remark 1.7. For convenience, one may consider Turing machines whose initial configuration is not of the form $[q_0]$ but is any fixed configuration c_0. This does not add any expressive power, as can be proved by a simple encoding of c_0 into the control state set of the machine, which will not be detailed here. This remark still holds for the sub-families of Turing machines presented in the next sections.

1.3.2.3 Languages

As previously mentioned, the languages accepted by Turing machines are called recursively enumerable. This name precisely comes from the fact that these sets of words are those Turing machines can enumerate. However, it is impossible to test the membership of a word in some recursively enumerable sets: any Turing machine accepting such a language L will necessarily fail to halt on some words which do not belong to L, making it impossible to know during a given run

whether the machine will eventually stop and accept its input word, or not. It is thus interesting to also distinguish a *strict* sub-class of this family, called recursive languages, which are the languages of Turing machines with no infinite run on any input word. Recursive languages play an important role in decidability theory. A problem whose positive instances can be encoded as a recursive language is called *decidable*. An intuitive reformulation of the sentence 'problem P is decidable' is: 'there exists an algorithm to solve P which halts on any input'.

A well-known property of recursively enumerable languages is that they are not closed under complement, but that any recursively enumerable language whose complement is also recursively enumerable is in fact recursive. The proof of existence of non recursively enumerable languages requires diagonalization arguments (see for instance [HU79] for more details).

We will now provide the definitions of three other families of acceptors for the three remaining classes of languages in the Chomsky hierarchy. All are defined as syntactical restrictions of Turing machines.

1.3.3 Linearly Bounded Machines

When only allowed to use at most as many cells as the number of input letters they have read, Turing machines accept precisely the type 1 (context-sensitive) languages. In that case, they are called linearly bounded machines. However, it is undecidable whether a given Turing machine is linearly bounded. For this reason, we provide a syntactic restriction of Turing machines rules such that the obtained machines are linearly bounded and accept all context-sensitive languages.

Definition 1.8. A linearly bounded machine (LBM) is a Turing machine $M = (\Sigma, \Gamma, [,], Q, q_0, F, \delta)$, where no insertion rule $pB \xrightarrow{a} qAB \in \delta$ is labeled by ε.

One can very easily see this restriction implies that such a machine uses at most $|w|$ tape cells during any run on a word w. Indeed, each time the machine allocates a new cell using an insertion rule, it also has to read an input letter.

Example 1.9. The linearly bounded Turing machine working on input and tape alphabet $\{a, b\}$ with control states $\{q_0, q_1, q_2, q_3\}$, with q_0 initial and q_2 the unique terminal state, and whose transition rules are

$$q_0] \xrightarrow{a} q_0a] \qquad q_1a \xrightarrow{b} q_1b+ \qquad q_2b \xrightarrow{a} q_3a- \qquad q_3b \xrightarrow{a} q_3a-$$
$$q_0a \xrightarrow{a} q_0aa \qquad q_1] \xrightarrow{\varepsilon} q_2]- \qquad\qquad\qquad q_3[\xrightarrow{\varepsilon} q_1[+$$
$$q_0a \xrightarrow{b} q_1b+$$

accepts the context-sensitive language $\{(a^nb^n)^+ \mid n \geq 1\}$. This machine is deterministic because no pair of rules are applicable from the same configuration. Its

unique run on word *aabbaabb* is

$$[q_0] \xrightarrow{a} [q_0 a] \xrightarrow{a} [q_0 aa] \xrightarrow{b} [bq_1 a] \xrightarrow{b} [bbq_1] \dots$$
$$\dots \xrightarrow{\varepsilon} [bq_2 b] \xrightarrow{a} [q_3 ba] \xrightarrow{a} q_3[aa] \xrightarrow{\varepsilon} [q_1 aa] \xrightarrow{b} [bq_1 a] \xrightarrow{b} [bbq_1] \xrightarrow{\varepsilon} [bq_2 b],$$

after which the machine can either accept this word (since q_2 is terminal) or repeat the cycle $[bq_2 b] \xrightarrow{aabb} [bq_2 b]$ any number of times.

The class of type 1 or context-sensitive languages is strictly included in that of recursive languages. Consequently, it is decidable to determine whether any given linearly bounded machine accepts a given word, and also whether a linearly bounded machine has an infinite run on any input word. In fact, one can show that any linearly bounded machine can be transformed in such a way that it has no infinite run on any input word.

Proposition 1.10. *For all linearly bounded Turing machine, there exists a terminating linearly bounded Turing machine recognizing the same language.*

Proof. It suffices to show that, for all linearly bounded machine M whose set of transitions contains a cycle, there is an equivalent terminating machine M', i.e. a machine which always terminates and accepts the same language as M. The total number of distinct configurations of M during a run on a word of size n is bounded by k^n, where k is a constant depending on the size of the control state set and work alphabet of M. It is easy to see that a word w accepted by M must be accepted by at least one run of size less than $k^{|w|}$. Indeed, if the smallest run accepting w was longer than this bound, it would necessarily contain two occurrences of the exact same configuration, i.e. a cycle. By removing this cycle, one would obtain a shorter accepting run. Let M' be the machine which simulates M while incrementing a n-digit counter in base k encoded in the alphabet, and stops the run if the counter overflows. By construction, M' would accept the very same language as M, without ever running for more than $k^{|w|}$ steps. Similar arguments are used in [Kur64] to show that the set of words on which a linearly bounded machine has an infinite run is context-sensitive. □

Unlike recursively enumerable languages, context-sensitive languages form an effective Boolean algebra: they are closed under union and complement.

Theorem 1.11 ([Imm88])**.** *The context-sensitive languages over a finite alphabet* Γ *form an effective Boolean algebra.*

The notion of space-bounded Turing machine can be extended to any bound $f : \mathbb{N} \mapsto \mathbb{N}$ such that for all $n \geq 0$, $f(n) \geq n$. The set of all languages accepted by a Turing machine working in space $f(n)$ on an entry of size n is written NSPACE$[f(n)]$. The following theorem states that the space hierarchy is strict.

Theorem 1.12 ([HU79, Imm88])**.** *For all space-constructible f and g in $\mathbb{N} \mapsto \mathbb{N}$ such that $\lim_{n\to+\infty} \frac{f(n)}{g(n)} = 0$, we have* $\mathrm{NSPACE}[f(n)] \subsetneq \mathrm{NSPACE}[g(n)]$.

In particular, the family $\mathrm{NSPACE}[2^n]$ of languages recognizable in exponential space strictly contains the family of context-sensitive languages ($\mathrm{NSPACE}[n]$).

Since they are accepted by a syntactic restriction of Turing machines, every context-sensitive language is recursively enumerable (and even recursive, thanks to their closure under complement). The existence of recursive languages which are not context-sensitive requires diagonalization arguments. Unlike recursively enumerable, we do not know whether all context-sensitive languages are accepted by deterministic linearly bounded Turing machines. This question was first raised by Kuroda in [Kur64].

1.3.4 Pushdown and Higher-Order Pushdown Automata

At the next level of the Chomsky hierarchy lie the context-free languages. This family is characterized by the very well-known class of pushdown automata, which additionally to their finite control use an auxiliary stack-like memory. On each transition, they can consult the top-most symbol of their stack, and either discard it or replace it with new symbols on top of the stack. As we did for linearly bounded machines, we will present pushdown automata as a syntactic restriction of Turing machines. The stack will be represented by the tape content, with the convention that leftmost cells represent elements which are higher up on the stack.

Definition 1.13. A pushdown automaton is a Turing machine $P = (\Sigma, \Gamma, [,], Q, q_0, F, \delta)$ with no rules of type 1.1 (cf. Def. 1.6).

This restriction ensures that the read head of a pushdown automaton never moves along the tape, and always faces the second leftmost symbol (the first one being the left boundary symbol). All configurations reachable by P from the initial configuration $[q_0]$ are indeed of the form $[qs]$, with $q \in Q$ and $s \in \Gamma^*$, which we abbreviate as simply qs. When dealing with pushdown automata, the tape will be called a *stack*. Insertion rules are seen as the addition of a new symbol on top of the stack, informally called a *push* operation, and deletion rules are seen as removing a symbol from the top of the stack (*pop* operation). Rewriting rules are equivalent to a *pop* followed by a *push*.

Example 1.14. The pushdown automaton working on input alphabet $\{a, b\}$ and stack alphabet $\{a\}$ with control states $\{q_0, q_1, q_2\}$, with q_0 initial and q_2 the unique

terminal state, and whose transition rules are

$$q_0] \xrightarrow{a} q_0 a] \qquad q_0 a \xrightarrow{b} q_1 \qquad q_1] \xrightarrow{\varepsilon} q_2]$$
$$q_0 a \xrightarrow{a} q_0 aa \qquad q_1 a \xrightarrow{b} q_1$$

accepts the context-free language $\{a^n b^n \mid n \geq 1\}$. This machine is deterministic because no pair of rules are applicable from the same configuration. Its unique run on word $a^n b^n$ is the accepting run

$$[q_0] \xrightarrow{a} [q_0 a] \xrightarrow{a} [q_0 aa] \xrightarrow{a} \dots$$
$$\dots \xrightarrow{a} [q_0 a^n] \xrightarrow{b} [q_1 a^{n-1}] \xrightarrow{b} [q_1 a^{n-2}] \xrightarrow{b} \dots$$
$$\dots \xrightarrow{b} [q_1] \xrightarrow{\varepsilon} [q_2].$$

In some definitions of pushdown automata, general rules of the form $pA \xrightarrow{a} qW$ where $W \in \Gamma^*$ are allowed. Such rules allow, in a single step, to pop the top-most A symbol from the stack and then push all the letters of W in reverse order (i.e. starting from the last letter). Also, it is sometimes required that only when the stack is empty can a configuration be accepting.

These alternative definitions do in fact not change the expressive power of the model, and simple constructions allow to simulate these different formalisms using the present one, potentially at the cost of a larger set of ε-rules or a larger stack alphabet[5]. Unlike context-sensitive languages, context-free languages are neither closed under complement nor intersection, and unlike Turing machines, pushdown automata can not always be determinized: some context-free languages are accepted by no deterministic pushdown automaton.

Context-free languages are strictly included in context-sensitive languages. For instance, the languages $\{a^n b^n c^n \mid n \in \mathbb{N}\}$ or $\{(a^n b^n)^+ \mid n \geq 1\}$ from Example 1.9 are context-sensitive but not context-free.

Higher-Order Pushdown Automata

Between the classes of context-free and context-sensitive languages lies a strict hierarchy of languages called the OI-hierarchy [Dam82]. They were characterized as the languages accepted by *higher-order pushdown automata*. In addition to manipulating single symbols, these automata are able to duplicate or destroy whole stacks of symbols, stacks of stacks, and so on. Before presenting the formal definition of higher-order pushdown automata, we define higher-order pushdown stacks and operations.

[5]To be accurate, acceptation by empty stack also requires to only consider languages which do not contain the empty word ε.

A *level 1 stack* over alphabet Γ is a word $w \in \Gamma^*$, written $[w]$. The empty level 1 stack is written $[\,]$. For $n \geq 2$, a *level n stack* over Γ is a sequence $[s_1 \dots s_k]$ of level $n-1$ stacks. The set of higher-order stacks of level $n \geq 1$ is written S_n. The level of a stack s is written $l(s)$. The following operations are defined on level 1 stacks:

$$\begin{aligned} push_1^a([a_1 \dots a_l]) &= [a a_1 a_2 \dots a_l] && \text{for all } a \in \Gamma, \\ pop_1([a_1 \dots a_l]) &= [a_2 \dots a_l] && \text{for } l \geq 1, \end{aligned}$$

The following operations are defined on level n stacks ($n > 1$):

$$\begin{aligned} push_1^a([s_1 \dots s_l]) &= [push_1^a(s_1) \dots s_l] && \text{for all } a \in \Gamma, \\ push_k([s_1 \dots s_l]) &= [push_k(s_1) \dots s_l] && \text{for } k \in [2, n[, \\ push_n([s_1 \dots s_l]) &= [s_1 s_1 \dots s_l] \\ pop_k([s_1 \dots s_l]) &= [pop_k(s_1) \dots s_l] && \text{for } k \in [1, n[, \\ pop_n([s_1 \dots s_l]) &= [s_2 \dots s_l] && \text{for } l > 1, \text{ otherwise undefined.} \end{aligned}$$

We also define a function *top* allowing to inspect the top-most symbol in the top-most level 1 stack as $top_1([a_1 \dots a_l]) = a_1$ for $[a_1 \dots a_l] \in S_1$, $top_1([\,]) = \varepsilon$ and $top([s_1 \dots s_l]) = top(s_1)$ for $[s_1 \dots s_l]$ of level greater than 1. We denote by O_n the set of all operations defined over level n stacks. We say that operation o is of level n, written $\mathrm{l}(o) = n$, if o is either $push_n$ or pop_n, or $push_1^a, pop_1$ if $n = 1$. We can define higher-order pushdown automata.

Definition 1.15. A higher-order pushdown automaton of level n is a tuple $P = (\Sigma, \Gamma, [,], Q, q_0, F, \delta)$, where Σ and Γ are finite sets of *input* and *stack symbols* respectively, $[$ and $] \notin \Gamma$ are *stack delimiters*, Q is a finite set of *control states*, $q_0 \in Q$ is the unique *initial state*, $F \subseteq Q$ is a set of *final states* and δ is a finite set of *transition rules* of the form $pA \xrightarrow{l} qo$ with $p, q \in Q$, $A \in \Gamma \cup \{\varepsilon\}$, $o \in O_n$ and $l \in \Sigma \cup \{\varepsilon\}$.

The level of a transition rule $pA \xrightarrow{l} qo$ is simply $\mathrm{l}(o)$. Configurations of a level n higher-order pushdown automaton P are pairs (q, s), where $q \in Q$ and s is a level n stack over Γ. Transition rules of P induce a labeled transition relation $(\xrightarrow[P]{l})_{l \in \Sigma \cup \{\varepsilon\}}$ over its set of configurations such that $(p, s) \xrightarrow[P]{l} (q, s')$ if and only if $top(s) = A$, P has a rule $pA \xrightarrow{l} qo$ and $s' = o(s)$. Runs, determinism, and accepted languages of higher-order pushdown automata are defined as usual.

Example 1.16. The level 2 pushdown automaton working on input alphabet $\{a, b, c\}$ and stack alphabet $\{a\}$ with control states $\{q_0, q_1, q_2, q_3\}$, with q_0 initial

and q_3 the unique terminal state, and whose transition rules are

$$q_0\varepsilon \xrightarrow{a} q_0 push_1^a \qquad q_1 a \xrightarrow{b} q_1 pop_1 \qquad q_2 a \xrightarrow{c} q_2 pop_1$$
$$q_0 a \xrightarrow{a} q_0 push_1^a \qquad q_1\varepsilon \xrightarrow{\varepsilon} q_2 pop_2 \qquad q_2\varepsilon \xrightarrow{\varepsilon} q_3 push_2$$
$$q_0 a \xrightarrow{\varepsilon} q_1 push_2$$

accepts the language $\{a^n b^n c^n \mid n \geq 1\}$. Its unique run on word $a^n b^n c^n$ is the accepting run

$$\begin{aligned}(q_0, [[\,]]) &\xrightarrow{a} (q_0, [[a]]) \xrightarrow{a} (q_0, [[aa]]) \xrightarrow{a} \dots \\ &\dots \xrightarrow{a} (q_0, [[a^n]]) \xrightarrow{\varepsilon} (q_1, [[a^n][a^n]]) \xrightarrow{b} (q_1, [[a^{n-1}][a^n]]) \xrightarrow{b} \dots \\ &\dots \xrightarrow{b} (q_1, [[\,][a^n]]) \xrightarrow{\varepsilon} (q_2, [[a^n]]) \xrightarrow{c} (q_2, [[a^{n-1}]]) \xrightarrow{c} \dots \\ &\dots \xrightarrow{c} (q_2, [[a]]) \xrightarrow{c} (q_2, [[\,]]) \xrightarrow{\varepsilon} (q_3, [[\,][\,]]).\end{aligned}$$

Level 1 automata are obviously equal to standard pushdown automata, and accept the context-free languages. The languages accepted at each level form a strict hierarchy, while remaining included in context-sensitive languages. Examples of non context-free languages accepted by level 2 higher-order pushdown automata are $\{ww \mid w \in \Sigma^*\}$ or $\{a^n b^n c^n\}$ for instance. For more details about the OI hierarchy, see [Dam82, Eng83].

1.3.5 Finite Automata and Related Notions

We now present three of the alternative characterizations of type 3 languages, beginning with finite automata and their languages. The study of finite automata and their variants is a very rich topic with links to many different areas of computer science, from program verification to the algebraic study of semigroups. For an introduction to the basics of the theory, the interested reader is referred to [HU79, Sak03]. Applications of finite automata are too many to enumerate. Section 1.6 mentions some aspects of program verification methods based on automata theory.

Finite automata, like linearly bounded machines and pushdown automata, can be defined as a straightforward restriction of our definition of Turing machines. In fact, finite automata only retain the finite memory provided by their set of control states, and can not access the auxiliary tape at all.

Definition 1.17. A finite automaton $A = (\Sigma, \Gamma, [,], Q, q_0, F, \delta)$ is a Turing machine with only rules of the form $p] \xrightarrow{l} q]$ where $p, q \in Q$ and $l \in \Sigma \cup \{\varepsilon\}$.

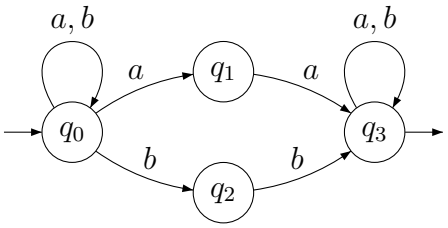
Due to this restriction, configurations reachable from $[q_0]$ in a finite automaton are always of the form $[q]$, simply abbreviated as q, where q is a control state.

Obviously, finite automata owe their name to the fact that they can only reach a finite number of configurations. A transition rule $p] \xrightarrow{a} q]$ will also be abbreviated as $p \xrightarrow{a} q$. Since finite automata do not use the tape, we will in general omit $\Gamma, [$ and $]$ altogether, and simply write $A = (\Sigma, Q, q_0, F, \delta)$. Finite automata are most often represented as state-transition diagrams, i.e. graphs whose vertices represent control states and whose edges represent transitions. Initial and terminal control states are respectively marked with an incoming and outgoing arrow.

Example 1.18. Automaton $A = (\{a, b\}, \{q_0, q_1, q_2, q_3\}, q_0, q_1, \delta)$ with transitions

$$\delta = \{q_0 \xrightarrow{a} q_0,\ q_0 \xrightarrow{b} q_0,\ q_0 \xrightarrow{a} q_1,\ q_0 \xrightarrow{b} q_2,\\ q_1 \xrightarrow{a} q_3,\ q_2 \xrightarrow{b} q_2,\ q_3 \xrightarrow{a} q_3,\ q_3 \xrightarrow{b} q_3\}$$

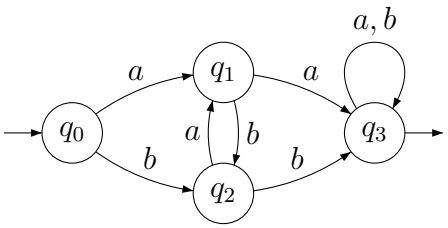
is represented by the diagram



The language $L(A)$ of this automaton is the set of words over $\{a, b\}$ containing at least one occurrence of one of the factors aa and bb.

The class of languages accepted by finite automata is precisely the class of type 3 languages in the Chomsky hierarchy, usually referred to as regular languages. Unlike pushdown automata, it can be shown that any finite automaton can be transformed into a (minimal) deterministic one with no ε-transitions.

Example 1.19. The deterministic automaton represented by the following diagram accepts the same language as the one from Ex. 1.18. It is the minimal automaton associated to this language.



Regular languages are closed under many operations over languages, including union and complement (and thus also intersection). They are strictly included in the family of context-sensitive languages (for instance, the language $\{a^n b^n \mid n \in \mathbb{N}\}$ is context-free but nor regular).

Finite automata and their languages constitute one of the most fundamental families in theoretical computer science, and even computer science in general. Several other characterizations of these languages exist, in particular as the rational or recognizable subsets of Σ^*.

1.3.5.1 Recognizable and rational subsets of a monoid.

Recognizability and rationality are algebraic notions defined over monoids. The former uses to notion of monoid morphism, and the latter uses the so-called 'Kleene star' operation over monoids.

A (homo)morphism from a monoid M to a monoid N is defined as a mapping ϕ from M to N such that $\phi(a \cdot b) = \phi(a) \cdot \phi(b)$ for all $a, b \in M$, and $\phi(1_M) = 1_N$. As a consequence, for all set S of generators of M, ϕ is fully defined by its restriction to S. In the special case where M is the free monoid Σ^*, it is sufficient to specify $\phi(a)$ for all $a \in \Sigma$ to define ϕ. A *recognizable* subset R of a monoid M is any subset of M for which there exists a *finite range* morphism ϕ such that $\phi^{-1}(\phi(R)) = R$.

We now define the Kleene star operation over a monoid M. Let S be any subset of M, we define S^0 as the singleton $\{\varepsilon\}$, and S^i inductively as $S \cdot S^{i-1}$ for all $i > 0$, where $\cdot$ denotes the product operation of M extended to sets. The Kleene star of S is defined as $S^* = \bigcup_{i \geq 0} S^i$. We also sometimes use the notation S^+ for the set $\bigcup_{i>0} S^i$. The set of *rational* subsets of M is the smallest family containing the finite sets and closed under union, product and Kleene star. In other words, a subset of M is rational if it can be expressed using finite sets and the set union, product and Kleene star operators. Rational sets are thus often described using expressions over the alphabet symbols and these operators, called regular or rational expressions.

One important result of computer science is the fact that recognizable and rational subsets of the free monoid both coincide with the set of languages accepted by finite automata [Kle56], and with the languages defined by monadic second-order formulas [Büc62] (cf. Section 1.5.2), and that algorithms exist to perform all translations from one formalism to the other. See [Wei04] for a clear and concise summary, and for instance [Sak03] for a more detailed exposition. The equality between rational and recognizable sets does not hold for any finitely generated monoid: in general, recognizable sets are rational, but the converse may not hold [Kni64] (this is the case for the monoid of binary relations over words for instance, cf. Section 1.4).

1.3.5.2 Finite Tree Automata

Following the development of language theory over words, notions of languages and automata generalized to the framework of sets of finite terms and trees emerged (see [CDG$^+$] for an introduction to the field). Despite recent progress [EW05], the knowledge of algebraic structures underlying terms and trees is far less developed than that of the free monoid, and one cannot yet speak of a full-fledged algebraic theory of term languages. However, a few notions became accepted in the community as the most natural extensions of the word case, and in particular the equivalent characterization by logic, automata recognizability, generalized rational expressions and algebraic recognizability of *regular* sets of terms. We will not detail all elements of this equivalence, but focus on finite tree automata which will be used extensively in this document.

Finite tree automata run on finite trees representing ground terms over a ranked alphabet F. They are similar to their word counterpart in the sense that they evaluate their input symbol by symbol while maintaining a finite memory using their finite set of control states. However, due to the richer topology of trees, finite tree automata have to run on several branches of their input at once. This is usually done in one of two ways: either the tree is evaluated starting from the leaves towards the root ('bottom-up'), or from the root towards the leaves ('top-down').

Bottom-up tree automata.

Definition 1.20. A finite bottom-up tree automaton is a tuple $A = (F, Q, Q_f, \delta)$ where F is a ranked input alphabet, Q a set of control states, $Q_f \subseteq Q$ is a set of final states and δ is a set of transition rules of the form $f(q_1, \ldots, q_n) \to q$.

During their runs, finite automata over words intuitively associate a control state to each prefix of their input word, namely the control state they reach after that prefix has been read. Similarly, bottom-up tree automata associate a control state to any ground sub-term they read as input. Each rule $f(q_1, \ldots, q_n) \to q$ of a bottom-up tree automaton A states that a term $ft_1 \ldots t_n$ such that each t_i can be associated by A to control state q_i can itself be associated with control state q[6]. More formally, the transition relation $\underset{A}{\longrightarrow}$ of A is defined as the derivation relation of the finite linear term rewriting system

$$\{(fq_1x_1 \ldots q_nx_n, qfx_1 \ldots x_n) \mid f(q_1, \ldots, q_n) \to q \in \delta\}$$

over $T(F \cup Q)$, where elements of Q are considered unary. A run of A over input term t is a derivation sequence of this system starting from t. A term t is

[6]Note that in the specific case where f is a constant, say a, the form of a rule becomes $a \to q$, meaning that a single leaf labeled by a can be associated to control state q

accepted by A if it can be derived into term $q_f t$, for some terminal state $q_f \in Q_f$. As usual, A is called deterministic if it has at most one run on any input term (or, equivalently, at most one rule with the same left-hand side), and non-ambiguous if it has at most one accepting run per term.

Example 1.21. The deterministic bottom-up tree automaton over alphabet $\{f^{(2)}, g^{(1)}, a^{(0)}, b^{(0)}\}$ with final state q_2 and the set of transition rules

$$\begin{array}{lll} a \to q_0 & gq_0 \to q_0 & fq_0q_1 \to q_2 \\ b \to q_1 & gq_1 \to q_1 & fq_1q_0 \to q_2 \end{array}$$

accepts the set of terms $\{f(g^*a, g^*b)\} \cup \{f(g^*b, g^*a)\}$. A run over input term $f(gga, gb)$ can be described as the derivation sequence

$$\begin{array}{c} f(gga, gb) \longrightarrow f(gg(q_0a), gb) \longrightarrow f(g(q_0ga), g(q_1b)) \\ \longrightarrow f(q_0gga, q_1gb) \longrightarrow q_2(f(gga, gb)). \end{array}$$

Top-down tree automata.

Definition 1.22. A finite (top-down) tree automaton is a tuple $A = (F, Q, Q_i, \delta)$ where F is a ranked input alphabet, Q a set of control states, $Q_i \subseteq Q$ is a set of initial states and δ is a set of transition rules of the form $q \to f(q_1, \ldots, q_n)$.

The semantics of top-down automata are precisely dual to those of bottom-up ones, and the transition relation of A is defined as the inverse of that of the corresponding bottom-up automaton $A' = (F, Q, Q_i, \delta^{-1})$ whose set of terminal states is Q_i and whose rules are the rules of A with the arrow reversed. Consequently, runs of A are reversed runs of A', and a run of A is accepting if it is a reversed accepting run of A'. Hence obviously, bottom-up and top-down automata recognize the same family of languages, which we will refer to as regular term languages from this point on. This family is closed under boolean operations and linear and inverse homomorphisms.

The only difference between both variants of tree automata concerns determinism. In general, the top-down tree automaton corresponding to a deterministic bottom-up one is itself not deterministic. Deterministic and non-deterministic bottom-up automata have the same expressive power, but this is not the case for top-down automata, whose deterministic versions are strictly less expressive than their non-deterministic counterparts. For instance, the language accepted by the deterministic automaton from Example 1.21 is accepted by a non-deterministic top-down automata, but by no deterministic one.

Partial runs. We introduce special terminology concerning runs of tree automata on term words. We say a ground term word t is *accepted* by a top-down automaton A if each term composing it is accepted by A. We say an n-context t is *partially accepted* by A from control state q to control word $q_1 \dots q_n$ if there is a run of the automaton from configuration $q(t)$ to configuration $t[q_1(\Box_1) \dots q_n(\Box_n)]$. A k'-contextual term word $t_1 \dots t_k$ of length k is partially accepted by A from control word $q_1 \dots q_k$ to control word $q'_1 \dots q'_{k'}$ if each t_i is partially accepted by A from state q_i to control word u_i and $u_1 \dots u_k = q'_1 \dots q'_{k'}$. Words $q_1 \dots q_k$ and $q'_1 \dots q'_{k'}$ are respectively called the *initial* and *final* control words of the run. This terminology extends to bottom-up automata by duality.

1.3.6 A Few More Remarks

The four families of acceptors we just presented indeed form a hierarchy, as can be seen by the successive definitions of linearly bounded machines, pushdown automata and finite automata as increasingly restricted Turing machines. The languages they accept form the Chomsky hierarchy, which is strict.

Automata and machines over more general (in particular non-free) monoids may be defined by replacing the input alphabet Σ in the previous definitions by any finite subset of any monoid, and the label ε by the neutral element of that monoid. Along a run, one would no longer consider concatenations of input symbols, but rather their product in the monoid, to define the set of elements accepted by the machine. An example of a class of finite automata over a more general monoid is the class of finite transducers, presented in Section 1.4.2.

To end this section, we would finally like to stress the fact that each of the four families of finite acceptors presented above can be seen as a family of word rewriting systems over a certain domain. This is not a surprising fact, since we used rewriting systems to define Turing machine transition relations, but keeping this in mind will have some importance for some of the material presented here (cf. Sect 2.1.1).

1.4 Families of Binary Relations

This section presents some of the most well-known families of binary relations over words. We consider sets of pairs of words seen as elements of the product monoid $\Sigma^* \times \Sigma^*$, whose composition law is defined as $(u_1, v_1) \cdot (u_2, v_2) = (u_1u_2, v_1v_2)$. It is natural to consider recognizable and rational subsets of this monoid, which we will call respectively recognizable and rational relations over words. Note that $\Sigma^* \times \Sigma^*$ is finitely generated by the set $(\Sigma \times \{\varepsilon\}) \cup (\{\varepsilon\} \times \Sigma)$ but is not free, and the families of recognizable and rational sets do not coincide as in the case

of words. We present both families using automata, and also a few interesting families of relations lying in-between them. To end this section, we briefly discuss possible extensions of each of these notions to terms.

1.4.1 Recognizable Relations (and some Extensions)

Recognizable sets within a direct product of two monoids have the following interesting property, usually called Mezei's theorem: given two monoids M and N, the recognizable parts of the product monoid $M \times N$ can be seen as a finite unions of products $U_i \times V_i$, where each U_i and V_i is a recognizable subset of the corresponding monoid. In the case of binary relations over words, this means every recognizable relation over $\Sigma^* \times \Sigma^*$ can be seen as a finite union of products $K_i \times L_i$, where all K_i and L_i are regular.

Following this result, one can characterize a recognizable relation R over words in Σ^* using for instance a finite set of pairs (A_i, B_i) of finite automata over Σ. We say a pair of words (u, v) is in R if $u \in L(A_i)$ and $v \in L(B_i)$ for some i.

Example 1.23. The binary relation from $\{a\}^*$ to $\{b\}^*$ associating any word of even length to a word of odd length (and conversely) is recognizable since it can be written $\big((a^2)^* \times (b^2)^*b\big) \cup \big((a^2)^*a \times (b^2)^*\big)$.

Prefix-Recognizable Relations

Recognizable relations are very weak, and do not allow any form of synchronization between a word and its image. For instance, the identity relation is not recognizable. A slight extension allowing more interesting relations to be specified is to consider the right (or left) *rational closures* of recognizable relations [Cho82]. A relation R' is the rational right closure of a relation $R = \bigcup_{i\in[1,n]}(U_i \times V_i)$ if it is defined as a finite union of sets of pairs $\bigcup_{i\in[1,n]}\{(uw, vw) \mid (u, v) \in (U_i \times V_i), w \in W_i\}$, where each U_i, V_i and W_i is a regular language. One sometimes speaks of *prefix-recognizable* relations [Cau96, CC03]. These relations form a boolean algebra, and are closed under composition and transitive closure.

A special case is the class of rational closures by Σ^*, which we will call *simple closures* of recognizable relations. This class has less interesting properties than that of prefix-recognizable relations, since it is not closed under complement nor under intersection.

Example 1.24. The identity relation over alphabet Σ is the closure by Σ^* of the (recognizable) relation $\{\varepsilon\} \times \{\varepsilon\}$.
The relation $\{(a^n, b^n) \mid n \geq 0\}$ from $\{a\}^*$ to $\{b\}^*$ pairing words of the same length is neither recognizable nor the rational closure of a recognizable relation.
The relation $\{(a^n b^k, a^n c^l) \mid k, l, n \geq 0\}$ is the left closure by a^* of the recognizable relation $b^* \times c^*$.

1.4.2 Rational Relations (and some Restrictions)

We saw at the beginning of this section the definition of the product monoid's internal composition law. Union and Kleene star operations over subsets of $\Sigma^* \times \Sigma^*$ are defined in the usual way. Using these operations, one can then form rational expressions over this monoid denoting rational sets of pairs of words, which we will call rational relations or transductions.

As the rational subsets of a monoid, rational relations can also be characterized using finite automata over $\Sigma^* \times \Sigma^*$, called transducers. One usually considers transducers with labels in $(\Sigma \cup \{\varepsilon\}) \times (\Sigma \cup \{\varepsilon\})$. A transition labeled by (a, b) between control states p and q is written $p \xrightarrow{a/b} q$.

Example 1.25. The relation $\{(a^n, v) \mid n \geq 0, v \in b^*(ab^*)^n\}$ is a rational transduction, it can be represented by the rational expression $((\varepsilon, b)^*(a, a))^*(\varepsilon, b)^* = \{(a, a), (\varepsilon, b)\}^*$. It is also the relation accepted by a one-state transducer with two loops labeled by a/a and ε/b respectively.

We will not distinguish a transducer from the relation it accepts and write $(w, w') \in T$ if (w, w') is accepted by some transducer T. The *domain* $\mathrm{Dom}(T)$ and *range* $\mathrm{Ran}(T)$ of a transducer T are the sets $\{w \mid (w, w') \in T\}$ and $\{w' \mid (w, w') \in T\}$ respectively. A transducer recognizing a relation which is a function is called *functional*. A transducer is non-ambiguous if it has at most one accepting run on any pair (u, v).

1.4.2.1 Length-Preserving and Synchronized Relations

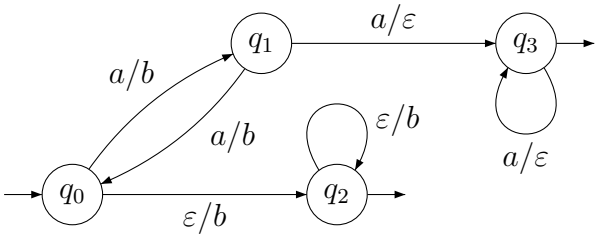
Interesting subclasses of rational relations are obtained when enforcing some form of synchronization between the ways transducer read their input and produce their output. In particular, rational relations over pairs of words of the same length, called *length-preserving* rational relations, coincide with the rational parts of the free monoid $(\Sigma \times \Sigma)^*$, and are thus recognized by finite automata with labels in $\Sigma \times \Sigma$, called *letter-to-letter* or *synchronous* transducers.

Example 1.26. The relation $\{(a^n, b^n) \mid n \geq 0\}$ is a length-preserving rational transduction. It is accepted by the one-state transducer with a single loop labeled by a/b.

A more relaxed form of synchronization introduced in [EM65] allows to use a special symbol $\Box$ to append at the end of shorter words so that they can be related to longer words using synchronous automata. These relations are usually called *automatic* or automata-definable. More formally, a relation R is automatic if and only if the relation $\{(u\Box^i, v\Box^j) \mid (u, v) \in R, i = \max(0, |v| - |u|), j = \max(0, |u| - |v|)\}$ is a length-preserving rational relation.

This class of relations was also studied in [FS93] under the name *synchronized relations*. A relation is left-synchronized if it can be written as a finite unions of products $R.S$, where R is length-preserving and S is of the form (L, ε) or (ε, L), with L a regular language over Σ. Equivalently, left-synchronized relations are the relations accepted by transducers in which for all run $q_0 \xrightarrow{x_0/y_0} q_1 \dots q_{n-1} \xrightarrow{x_n/y_n} q_n$, there exists $k \in [0, n]$ such that for all $i \in [0, k-1]$, x_i and y_i belong to Σ and either $x_k = \ldots = x_n = \varepsilon$ or $y_k = \ldots = y_n = \varepsilon$. Such transducers are called left-synchronized. Right-synchronized relations and transducers are defined similarly. Of course, synchronized transductions strictly include the rational closures of recognizable relations. They are however strictly less general than rational transductions (for instance, the relation of Example 1.25 is not accepted by any synchronized transducer).

Example 1.27. The relation $R = \{(a^{2k}, b^{m>2k}), (a^{2k+1}, b^{m\leq 2k}) \mid k \geq 0\}$ is left-synchronized. It is accepted by the following transducer:



Remark 1.28. If we apply the standard determinization procedure to a synchronous transducer, we obtain an equivalent non-ambiguous synchronous transducer (i.e for all pair of words (u, v) accepted by the transducer there is exactly one accepting run of the transducer labeled by u/v). This result is also true for synchronized transducers.

1.4.2.2 Sequential Relations

The standard notion of determinism for automata does not have much interest in the case of transducers because it does not rely only on the input but on both the input and the output. For instance, a transducer having transitions $q_0 \xrightarrow{a/b} q_1$ and $q_0 \xrightarrow{a/c} q_2$ would be considered deterministic, which does not fit the intuition. A more refined notion is that of sequentiality, which refers to the way a transducer consumes its input.

A transducer T with states Q is *sequential* if for all q, q' and q'' in Q, if $q \xrightarrow{x/y} q'$ and $q \xrightarrow{x'/y'} q''$ then either $x = x'$, $y = y'$ and $q' = q''$, or $x \neq \varepsilon$, $x' \neq \varepsilon$ and $x \neq x'$. In other words, a transducer is sequential if its underlying input automaton is deterministic. Note that sequential transducers are necessarily functional.

1.4.2.3 Properties of Rational Relations and Transducers

There is a close relationship between rational languages and rational relations. In particular, erasing the first (resp. second) component of every transition in a transducer yields a finite automaton recognizing the domain (resp. range) of the relation. Hence both the domain and range of a transducer are rational languages. Also, a fairly simple construction allows one to restrict the domain or the range of a transduction to a rational set, and this preserves the sequentiality or synchrony of the transducer.

Lemma 1.29. *Rational relations have rational domains and ranges, and are closed under restriction to a rational domain (resp. rational range). Moreover, let R be a rational language over Σ, for any sequential (resp. synchronous) transducer T over Σ, the transducer obtained by restricting T to the domain R is sequential (resp. synchronous).*

A very interesting consequence of these results is that the image of a regular language by a rational transduction is also regular: indeed, starting with such a language L, if one restricts the domain of a transducer T to L and then builds the automaton accepting the range of the obtained transducer, this automaton accepts precisely the language $T(L)$, which is thus regular. Rational transductions are said to *preserve* regularity. Since rational transductions are closed under converse, the inverse image of a regular language by such a relation is of course also regular.

Rational transductions are closed under composition and union, but they are not closed under complement or intersection. On the contrary, synchronized relations form a boolean algebra (they are closed under complement, union and intersection). They are in fact the largest known class of rational relations with this property. Also note that no class of relations at least as expressive as synchronous sequential transductions is closed under iteration, since they can very easily model deterministic Turing machine transitions.

1.4.3 Extensions to Terms

Most families of binary relations over words were extended with some success to binary relations over terms. However, similarly to the case of term languages, the situation is less clear from an algebraic point of view. In particular, there is no straightforward notion of rational relations for terms.

1.4.3.1 Recognizable Relations

By analogy with recognizable relations over words, one can define a recognizable term relation as the set of all pairs of terms accepted by a finite union of pairs

of finite tree automata. For instance, for $i \in [1, n]$, let (A_i, B_i) be a pair of finite non-deterministic top-down tree automata. The relation $R = \{(s, t) \mid \exists i \in [1, n], s \in L(A_i) \wedge t \in L(B_i)\}$ is recognizable. Recognizable relations share the same lack of expressiveness over terms as over words, and do not actually relate single terms but whole languages.

1.4.3.2 Ground Transductions

An interesting extension of recognizable relations generalizes right (or left) closures of recognizable word relations (cf. Section 1.4.1) to the case of terms. Such relations are called *ground tree transductions* and were defined in [DTHL87], together with a class of automata characterizing them, called *ground tree transducers* (GTT). A relation R is a ground tree transduction if it is of the form

$$R = \{(c[s_1 \dots s_n], c[t_1 \dots t_n]) \mid c \in C_n(F) \wedge \forall i \in [1, n], (s_i, t_i) \in R'\}$$

where R' is a recognizable relation. Ground transductions were initially defined to help study ground rewriting systems. Indeed, ground transductions coincide with the class of derivations of ground systems. They are closed under inverse, composition and transitive closure, but not under union or complement. This is reminiscent of the closures of recognizable relations over words. An interesting question would be to extend ground tree transductions in the spirit of *rational* closures of recognizable word relations, in the hope to obtain some of their properties, and in particular closure under boolean operations.

1.4.3.3 Synchronized Relations

Another class of relations which can be seen as a generalization from words is that of automatic relations (cf. Section 1.4.2). The same way automatic word relations use a special symbol $\Box$ to 'pad' the shorter of two words of different length before running a finite automaton on them, we will use the symbol $\perp \notin F$ to define the *overlap* of two terms t and s in $T(F)$, written $[ts]$. First, let $F_\times$ denote the ranked alphabet $\{fg \mid f, g \in (F \cup \{\perp\})\}$, where the arity of fg is the maximum of that of f and g. For $t = f(t_1 \dots t_n)$ and $s = g(s_1 \dots s_m)$, we define $[ts]$ inductively as

$$[f(t_1 \dots t_n), g(s_1 \dots s_m)] = fg([t_1 s_1] \dots [t_k s_k])$$

where $k = \max(n, m)$, $\forall i \in [n + 1, k], t_i = \perp$ and $\forall i \in [m + 1, k], s_i = \perp$. This operation is extended to pairs of sets of terms in the usual way. We will call *automatic* or *synchronized term relation* any binary term relation of the form $R = \{(s, t) \mid [st] \in L\}$, where L is a regular term language over $F_\times$. When restricted to words, this family of relations coincides with relations accepted by

synchronized transducers. It has the same property of being closed under boolean operations.

1.4.3.4 Rational Relations?

In our comparison between the word and tree binary relations, the last missing family is that of rational relations. Unfortunately, the situation with respect to rationality is far less clear for terms than for words. If rational relations over words enjoy a nice algebraic definition as the rational parts of a monoid, such a clear notion has yet to be found for terms. To paraphrase a survey of term relations by Raoult [Rao91], several criteria should be considered when proposing a candidate for the 'best' generalization:

1. When restricted to words, does the family considered coincide with rational relations? Does it at least contain the identity relation?
2. Is the class closed under composition? Under converse?
3. Is the class 'robust' (does it admit several equivalent natural definitions)?
4. Do relations of the class preserve regularity (i.e. map a regular set of terms to another regular set)? Do they have a regular domain and range?

Rational word relations answer yes to all of the above. Some of the questions asked in [Rao91] are omitted, but the point is that no existing class of term relations, up to now, matches all criteria to a satisfying level.

Despite this statement, several distinct families of relations attempting to generalize rational word relations were defined and brought many interesting results and applications. We refer the interested reader to the above-mentioned survey as well as [GS97, CDG$^+$] for a more exhaustive summary. In Chapter 3, we will use a notion of relations defined in [Rao97] which is more general than most common families of relations, but still retains interesting properties and is expressed using a very convenient grammar formalism.

1.5 Logics over Graphs

Logics are mathematical languages used to express certain properties of objects like words, terms, graphs, and more generally any kind of *relational structures*. We present two main kinds of logics over colored graphs, the well-known first-order logic and the monadic restriction of second-order logic, then present some of their variants. This section is obviously very superficial, its only aim being to fix notations and terminology.

1.5.1 First-Order Logic

Let $X = \{x, y, \ldots\}$ be a countable set of variables ranging over vertices. First-order logic formulas over Σ-labeled C-colored graphs are either:

- an atomic formula of the form *true*, $x = y$, $x \xrightarrow{a} y$ or $c(x)$ with $a \in \Sigma$, $c \in C$ and $x, y \in X$,
- a conjunction $\phi \wedge \psi$ of formulas ϕ and ψ or the negation $\neg\phi$ of a formula ϕ,
- an existential quantification $\exists x\phi$.

Given a colored graph (G, K), a *valuation* of X is a mapping associating a vertex $v \in V_G$ to each variable in X. Atomic formulas $x = y$, $x \xrightarrow{a} y$ and $c(x)$ hold in a graph (G, K) (or are *satisfied* by (G, K)) under valuation γ if $\gamma(x) = \gamma(y)$, $\gamma(x) \xrightarrow[G]{a} \gamma(y)$ and $K(\gamma(x)) = c$, respectively. A conjunction $\phi \wedge \psi$ holds if both ϕ and ψ do, and a negation $\neg\phi$ holds if ϕ does not. Finally, a formula $\exists x\phi$ holds if there exists a vertex $v \in V_G$ such that ϕ holds under valuation γ', where $\gamma'(y) = \gamma(y)$ for all $y \neq x$, and $\gamma'(x) = v$. Formula *true* always holds. The satisfaction of a formula ϕ in (G, K) under valuation γ is written $(G, K), \gamma \models \phi$.

Other usual connectives can be defined using this syntax: $\phi \vee \psi$ is equivalent to $\neg(\neg\phi \wedge \neg\psi)$, and $\forall x\phi$ is a shorthand notation for $\neg\exists\neg\phi$. In a formula of the form $\exists x\phi$ or $\forall x\phi$, all occurrences of variable x are called *bound*. Variables which have non-bound occurrences in a formula are *free*. We write $\phi(x_1, \ldots x_n)$ to denote the fact that at most variables x_1 to x_n occur free in ϕ. A formula in which all variable occurrences are bound is called a *sentence*. Satisfaction of a sentence ϕ by a graph (G, K) is simply noted $(G, K) \models \phi$, and we say (G, K) is a model for ϕ.

The set of models $L(\phi)$ of a given sentence ϕ is sometimes called the set *defined* by ϕ. In the case of graphs representing terms or words, one rather speaks of the *language* of ϕ. Such sets are referred to as *first-order definable*. Conversely, the set of all first-order sentences which are satisfied by a given graph (G, K) is called the *first-order theory* of (G, K), written $Th(G, K)$. The theory of (G, K) is said to be decidable if the set $Th(G, K)$ is recursive, i.e. if there exists an algorithm which decides whether a given formula holds on (G, K) or not.

First-order logic can basically express only 'local' properties of graphs. In particular, it is unable to express the existence of a path between two given vertices of a graph. Nevertheless, it has been shown [Tra50] that the first-order satisfiability problem is undecidable for finite graphs, i.e. it is undecidable whether a given first-order sentence admits at least one model in the set of finite graphs.

1.5.2 Monadic Second-Order Logic

We now present a very expressive logic over graphs called monadic second-order logic (MSO). Second-order logic is built similarly to first-order logic, with the

exception that quantification over variables denoting relations is allowed. The word monadic refers to the fact that variables in MSO only range over relations of arity 1, in other words sets.

Formally, monadic second-order logic extends the syntax of first-order logic with a countable set of higher-order variables $X, Y, X_1, \ldots$ denoting sets of graph vertices and new atomic predicates $x \in X$, where x a first-order variable and X a set variable. Quantification over set variables is now allowed, and valuations now range over both types of variables.

Intuitively, monadic second-order logic allows the description of path properties of graphs, like the existence of paths with a certain labels between two sets of vertices, the confluence of paths, etc.

1.5.3 Between FO and MSO

A common extension of first-order logic over Σ-labeled graphs is to extend its syntax with different types of reachability predicates. The weakest such extension allows a predicate of the form $x \xrightarrow{*} y$, which expresses the existence of a path between vertices denoted by x and y under the current interpretation of variables. More expressive extensions include predicates of the form $x \xrightarrow{A^*} y$, where A is a subset of the graph's label alphabet, or even $x \xrightarrow{L} y$ where L is a regular language over Σ. These last two types of predicates add the possibility to state the existence of a path with a certain label x and y, either by restricting the allowed label alphabet along this path, or by explicitly restricting to a regular set of path labels.

An even more general extension of first-order logic consists in adding a transitive closure operator TC in the syntax of the logic. Given a first-order formula $\phi(x, y)$ with free variables x and y, recursively define $TC(\phi)$ as $\exists z \phi(x, z) \wedge TC(\phi)(z, y)$. This new logic is strictly more expressive than the previously mentioned extensions, while remaining a subset of monadic second-order logic.

1.5.4 Temporal Logics

Temporal logics are modal logics specialized in the description of properties which refer to the passing of time in a system. When considering this kind of logics on graphs, time is supposed to increase by one unit whenever a transition is followed. Hence, temporal logics really express properties of the unfolding tree of a graph (cf Section 2.1.2.1) rather than the graph itself. All the logics we mention in this section can be easily expressed using monadic second-order formulas. As a matter of fact, all usual temporal logics are subsumed by a very expressive modal logic called μ-calculus, a fixed-point logic expressively equivalent to the bisimulation-invariant fragment of monadic second-order logic (see [JW96]), which we will not

present here.

1.5.4.1 CTL*

The logic CTL* (where CTL stands for Computation Tree Logic) is a logic able to express properties of several possible executions of a given system (see [Eme90]). Its formulas can be of two kinds: state formulas and path formulas. State formulas are built along the following syntax:

$$\phi = A\psi \mid E\psi \mid c \in C \mid \neg\phi \mid \phi_1 \wedge \phi_2$$

where C is the set of colors of the considered tree, ψ is a path formula and ϕ, ϕ_1, ϕ_2 are state formulas. Path formulas are built as follows:

$$\psi = \phi \mid X\psi \mid \psi_1 U \psi_2 \mid \neg\psi \mid \psi_1 \wedge \psi_2$$

where ϕ is a state formula (all state formulas are also path formulas) and ψ, ψ_1, ψ_2 are path formulas.

Well formed CTL* formulas are state formulas, which are evaluated at the root of a colored tree t. Atomic propositions (colors), hold if the root of the current tree (the 'present time' in the point of view of temporal logics) is labeled by this color. Quantifiers A or E range over the set of paths in the tree whose origin is the root. State formula $\mathsf{A}\phi$ and $\mathsf{E}\phi$ express that path formula ϕ holds with respect to all (resp. at least one) of the paths of the tree.

Path formulas are evaluated with respect to a given path π seen as a sequence of edges. We denote by π^i the path π deprived of its i first letters. A path formula ψ simply consisting of a state formula holds in t with respect to π if it holds in the subtree of t whose root is the origin of π. Formula $\mathsf{X}\psi$ (read *next* ψ) holds if ψ holds on π^1 (in other words, if ψ holds at the 'next time unit'). Formula $\psi_1 \mathsf{U} \psi_2$ (read ψ_1 *until* ψ_2) holds if there exists an integer $k \geq 0$ such that ψ_2 holds on π^k, and ψ_1 holds on π^j for all $j < k$ (in other terms ψ_2 will hold sometime in the future and ψ_1 has to hold continually until then).

Usual temporal operators F and G (respectively read *finally* or *eventually* and *globally* or *always*) can be expressed using the *until* operator. Path formulas $\mathsf{F}\phi$ and $\mathsf{G}\phi$ are respectively equivalent to $\mathit{true}\mathsf{U}\phi$ and $\neg\mathit{true}\mathsf{U}\neg\phi$.

1.5.4.2 LTL and CTL

Interesting (earlier) fragments of CTL* are LTL (linear temporal logic) [Pnu77] and CTL [CES86]. LTL formulas are defined as the set of formulas of the form $A\psi$ where ψ is a CTL* path formula. They are only able to speak about the general shape of all paths in a tree. The leading universal quantification is usually left implicit. CTL formulas are defined as formulas in which a strict alternation of

quantifiers and temporal operators is imposed. Both fragments are strictly less expressive than the whole logic and mutually incomparable.

1.6 Notions of Finite-State Model-Checking

The general model-checking problem, given a system S and a formula ϕ in any appropriate logic, is to decide whether $S \models \phi$, in other words whether the system verifies the given property. For instance, one may wish to determine, given a finite automaton modeling an abstraction of the behavior of a program and a LTL formula expressing a property of finite-state systems, whether the unfolding tree of this automaton satisfies that formula.

The general term 'model-checking' does not refer to a precise category of algorithms or techniques, but rather to the satisfaction problem itself. Details of existing techniques to solve this problem obviously vary a lot depending on the considered logic and class of models. However, in the case of finite systems (or their unfoldings) and the verification of their temporal properties, quite successful and wide-spread methods referred to as the automata-theoretic approach to verification exist.

In several cases, and in particular for CTL*, CTL and LTL formulas, the set of tree models of a given formula can be represented using certain classes of finite automata over infinite trees (or even infinite words when considering models of LTL as words). In that case, the model-checking problem simply translates into a membership test problem over these classes of automata. Other problems such as validity or satisfiability of a formula translate into universality and emptiness tests. We might want to ask more general questions of the kind: 'What is the *set* of models of formula ϕ?' or 'Do all elements of set S satisfy property ϕ, and if not, which elements do?'. Such problems directly translate into language intersection and inclusion problems.

The automata-theoretic approach of verification is a very rich subject, and is mostly out of scope of the present work (for more details, consult for instance [VW86, CES86]. The most important idea we want to stress at this point is that properties expressed using logics can be most of the time translated into automata whose languages are the sets of models of these properties.

Chapter 2

Infinite Graphs

As mentioned in the previous chapter, labeled graphs are finite sets of binary relations over a certain domain, where each vertex represents an element of the domain, and the relations between them are figured as edges. A consequence of this very general definition is that a very wide variety of formalisms exploiting binary relations can be expressed and represented as graphs.

In this chapter, we first present a variety of possible ways to represent families of infinite graphs in a finite manner, either by defining a set of representatives, in which case we speak of an internal representation, or by directly describing the structure of graphs without ever naming their vertices (external representations). In a second time we enumerate some of the most well-known families of infinite graphs, explicit their different internal and external representations and mention some of their key properties. Finally, we give a few hints about similarities between the study of infinite graphs and some recent trends in automatic verification of infinite systems.

2.1 Finite Presentations of Infinite Graphs

We first enumerate some of the ways in which families of infinite graphs can be described using finite formalisms. A distinction has to be made between two main approaches. The first way to describe a graph, which we will call an internal representation, is to directly specify a finite set of relations, one for each letter of the label alphabet, between elements of a certain domain, for instance the sets of words or terms on a certain alphabet.

Another category of presentations, referred to as external, directly describes the structure of graphs, either by characterizing them as the result of structural transformations from finite (or known) structures, as the sets of solutions of systems of equations ranging over graphs, as the sets of models of sentences in a given logical language, or as evaluations of sets of expressions describing graphs.

We give a brief overview of all these characterizations, with emphasis on internal representations, which are the most relevant to the present work.

2.1.1 Internal Presentations

As previously mentioned, a Σ-labeled graph can be defined simply as a labeled transition system, or in other words as a finite set of binary relations over the same domain, each relation being associated to a symbol in Σ. Hence conversely, any such set of relations is implicitly associated to a graph, and we do not distinguish both notions. In this setting, studying families of graphs (or rather graph representatives) amounts to studying families of binary relations.

Since we are only interested in finitely presented relations, and also in links between graphs and other areas of computer science (like language theory, rewriting theory and program verification), the easiest way to finitely characterize families of infinite graphs is using existing finite formalisms defining binary relations, and in particular machines (or automata), and rewriting systems.

2.1.1.1 Graphs Defined by Machines or Automata

In this section, given a finite alphabet Σ, we denote by M any acceptor (or 'machine') for words in Σ^* inducing a $(\Sigma \cup \{\varepsilon\})$-labeled transition relation $(\xrightarrow[M]{a})_{a\in(\Sigma\cup\{\varepsilon\})}$ over its set of configurations. M could be a Turing machine, a pushdown automaton, or virtually any other formalism. For simplicity, we just call M a *machine*, without any additional precision. We give a brief description of several standard ways such machines can be associated to infinite graphs.

Configuration graphs. The transition relation of a machine M can be directly seen as a labeled transition system over $\Sigma \cup \{\varepsilon\}$. We can in turn see this family of relations as a $(\Sigma \cup \{\varepsilon\})$-labeled graph, which we call the *configuration graphs* of M and write K_M.

The vertices of K_M are the configurations of M and its edges represent transitions of M from one configuration to another, including ε-transitions. This is the simplest way to associate a graph to a machine, and is basically the way finite graphs are used as representations for finite automata. As a matter of fact, the nuance between graphs and automata (or machines) only consists in the specification of initial and final configurations.

Transition graphs. The configuration graph of a machine M makes every transition appear as an edge. For most families of machines with an infinite set of configurations, configuration graphs may contain many edges labeled by ε. When considering these transitions as internal computation steps, one may wish

to conceal them and only consider the observable computational behavior of M from an external point of view. This can be done by transforming the graph K_M. However, when doing so, one should be careful not to introduce new path labels or destroy the global structure of K_M.

In the case of finite automata, we mentioned that any automaton containing ε-transitions can be transformed in an equivalent one, i.e. an automaton accepting the same language, not containing any ε-transitions. We generalize this idea to the configuration graphs of any machine M, and call the obtained result the *transition graph* of M.

Before stating the definition of transition graphs, we establish a distinction between *internal* and *external* configurations of a machine M, in the same way we distinguish internal transitions (ε-transitions) from external ones. A configuration c of M is called external if it is the source of at least one transition labeled by some $a \in \Sigma$, or if it is a sink (i.e. it has out-degree 0 in K_M). It is *internal* otherwise. We denote by C_e the set of external configurations of M.

Definition 2.1. Let M be word acceptor over Σ with ε-transitions, the *transition graph* of M is

$$G_M \;=\; \big\{(c_1, a, c_2) \mid c_1 \xrightarrow[M]{a\varepsilon^*} c_2,\ a \in \Sigma\ c_1, c_2 \in C_e\big\}.$$

For this definition to be meaningful, we need to assert a few extra hypotheses on the word acceptors we consider. In particular, since the purpose of transition graphs is to illustrate in a more synthetic way the structure of visible computations of machines, it is important that a transition graph be *faithful* to the corresponding machine and its configuration graph, in the sense that it should respect the reachability relation of the machine between configurations, and that all initial and final configurations should appear in the graph.

Definition 2.2. The transition graph G_M of a machine M over Σ is faithful if for all pair (c_1, c_2) of external configurations of M and for all $u \in \Sigma^*$, $c_1 \xrightarrow[K_M]{u} c_2 \implies c_1 \xrightarrow[G_M]{u} c_2$, and the initial configuration c_0 and all accepting configurations of M reachable from c_0 are vertices of G_M.

Figure 2.1 shows the configuration graph of a machine and the associated 'unfaithful' transition graph. A sufficient condition for the transition graph of a machine M to be faithful is for the external configurations of M to be the source of no ε-transition, and for the initial configuration and final configurations of M to all be external. We speak in this case of *normalized* machines. Intuitively, it means that at any point during computation, the machine can be in one of two modes: either it is in a stable mode, waiting for an input or reporting termination (either successful or unsuccessful), or it is in an unstable mode, processing

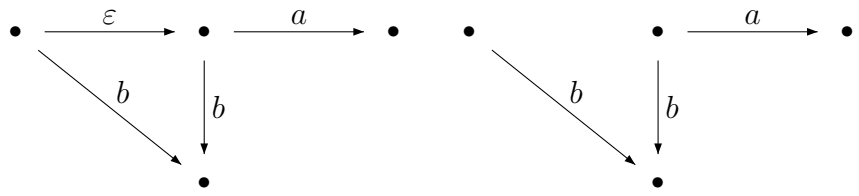


Figure 2.1: Configuration graph and associated (unfaithful) transition graph.

previous inputs by series of internal transitions. This notion was proposed by Stirling in [Sti00] to represent prefix-recognizable graphs as the transition graphs of pushdown automata. It was later used in [Car01] to generalize this result to the transition graphs of higher-order pushdown automata.

Proposition 2.3. *Transition graphs of normalized machines are faithful.*

Proof. Consider a normalized machine M over Σ, its configuration graph K_M and its transition graph G_M. Let (c_1, c_2) be any pair of external configurations of M such that $c_1 \xrightarrow[K_M]{u} c_2$ for some path label $u \in \Sigma^*$. Consider any path ρ labeled by u between c_1 and c_2 in the configuration graph K_M of M, let us prove by induction on the length of ρ that $c_1 \xrightarrow[G_M]{u} c_2$.

The case where ρ is the empty path is trivial because, since $u = \varepsilon$, $c_1 = c_2$. Suppose ρ is of length $n \geq 1$. If ρ starts with an ε-transition, this contradicts the fact that M is normalized and c_1 is an external configuration. Hence ρ must start with an a-transition with $a \in \Sigma$. Let c_3 be the first external configuration distinct from c_1 to be traversed by ρ. By definition of external configurations, the portion of path ρ between c_1 and c_3 must consist of an a-edge followed by any number of ε-edges. Hence by definition of transition graphs, there must be an edge $c_1 \xrightarrow{a} c_3$ in G_M. By induction on the remaining portion of ρ labeled by $a^{-1}u$ between c_3 and c_2, there must be a path $c_3 \xrightarrow{a^{-1}u} c_2$ in G_M, and thus $c_1 \xrightarrow[G_M]{u} c_2$. □

An interesting observation concerning transition graphs is that they only partially reflect the determinism of the machines defining them. First, note that a deterministic machine (i.e. a machine whose configuration graph is deterministic) necessarily has a deterministic transition graph. The converse is less satisfying: even non-deterministic machines may have deterministic transition graphs, as illustrated by Figure 2.2. We will see in Chapter 4 (Proposition 4.60) that normalized *terminating* machines (i.e. machines which do not have infinite non-accepting runs on any input word) can be determinized whenever their transition graph is deterministic.

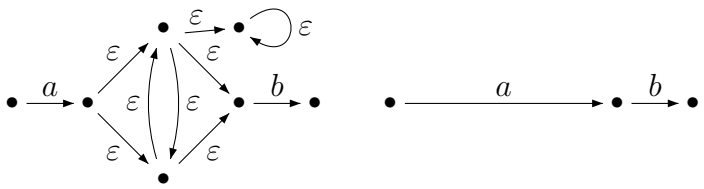


Figure 2.2: Configuration graph and deterministic transition graph of a non-deterministic machine.

Computation graphs. Finally, consider machines which, instead of accepting word languages, accept relations over an arbitrary domain V. Examples of such machines include rational transducers and Turing machines accepting pairs of words, for which V is the free monoid over some alphabet, or any kind of tree transducers, in which case V is a free algebra. We refer to such machines simply as *transducers*.

Let $(T_a)_{a\in\Sigma}$ be a finite family of transducers, and $L(T_a)$ denote the subset of $V \times V$ each T_a accepts. We define the computation graph of $(T_a)_{a\in\Sigma}$ as the graph in which all pairs (u, v) are bound by an a-edge whenever this pair is accepted by transducer T_a.

Definition 2.4. Let $T = (T_a)_{a\in\Sigma}$ be a finite family of transducers over domain V, we define the *computation graph* (or transduction graph) of T as

$$G_T = \{(u, a, v) \mid a \in \Sigma,\ (u, v) \in L(T_a)\}.$$

Computation graphs are not directly defined by the transition relations of machines as in the previous case. Rather, each of their edge relations is directly characterized by the sets of pairs accepted by a transducer. This last way to define graphs using finitely represented machines or automata may be the closest to the initial definition of a graph as a finite set of binary relations. However, it is not suited to model the same kind of situations, and their intuitive interpretation is a lot different from the intuition behind transition graphs, for instance.

The notion of *computation graph* was first introduced in early versions of [Cau03] and systematically used in [CK02a] (where it is called *relation graph*). These works prove that for pushdown automata and Turing machines, the classes of transition and computation graphs coincide.

2.1.1.2 Graphs Defined by Rewriting Systems

Any labeled word or term rewriting system naturally defines a graph whose edge relations are defined by the rewriting relation of the system. More precisely, we call *rewriting graph* of a Σ-labeled rewriting system R the graph $\{(u, a, v) \mid$

$u \xrightarrow[R]{a} v\}$. As a matter of fact, since transition relations of Turing machines are defined with respect to word rewriting systems (cf. Section 1.3.2), one can straightforwardly observe that the configuration graphs of all families of acceptors described in Section 1.3.1 are the rewriting graphs of the finite word rewriting systems describing their respective transition relations.

Another way to associate a graph to a given rewriting system was proposed in [CK98]. It relies on the derivation relations of word rewriting systems rather than their rewriting relations to define graphs with words as vertices. The *Cayley-type graph* of a word rewriting system is defined as follows.

Definition 2.5. The Σ-labeled Cayley-type graph of a word rewriting system R over Γ, with $\Sigma \subseteq \Gamma$, is the infinite graph

$$G_R \;=\; \{(u, a, v) \mid a \in \Sigma,\; u, v \in \mathrm{NF}(R),\; ua \xrightarrow[R]{*} v\}.$$

The terminology comes from an analogy with the group-theoretic concept of *Cayley graph* of a group. One can define a similar notion for term rewriting systems over some ranked alphabet F by considering Σ as a subset of the set F_1 of arity 1 symbols in F.

Definition 2.6. The Σ-labeled Cayley-type graph of a *term* rewriting system R over F, with $\Sigma \subseteq F_1$, is the infinite graph

$$G_R \;=\; \{(s, a, t) \mid a \in \Gamma,\; s, t \in \mathrm{NF}(R),\; a(s) \xrightarrow[R]{*} t\}.$$

An important difference between rewriting systems and language acceptors (or machines) is that the former lack the notions of inputs, internal and external transitions and configurations, runs, or acceptance. Therefore, it does not seem very meaningful to directly try and extend the notion of transition graphs to rewriting systems. In a way however, Cayley-type graphs bridge the gap between rewriting systems and machines by introducing notions of beginning and end of a derivation, stable configurations (normal forms), inputs (addition of a new symbol to a normal form) and termination (reaching a normal form).

2.1.1.3 Other Internal Characterizations

There are many other ways to give finite internal characterizations of infinite graphs. It should now be clear that any finite set of (labeled) binary relations can be seen as a graph. This is in particular true for all the families of binary relations presented in Section 1.4. We thus explicitly admit the expression *recognizable graph* for instance, as referring to finite unions of graphs of the form $\{(s, a, t) \mid s \in S, t \in T\}$, written $S \xrightarrow{a} T$, where S and T are regular sets over Γ and

$a \in \Sigma$. We also extend the operation of right (or left) closure to such graphs. The *rational right closure* of a recognizable graph $G = \bigcup_{i\in[1,n]}(S_i \xrightarrow{a_i} T_i)$ is a graph of the form $\bigcup_{i\in[1,n]}\{(su, a_i, tu) \mid s \in S_i, t \in T_i, u \in U_i\}$ where each U_i is a regular set. When all sets U_i are equal to Γ^*, one simply speaks of the right closure of G.

We already defined in the last chapter the restriction of a graph to a set of vertices. In the case where vertices are words (or terms), one may consider restricting the vertices of a graph G to a regular set L. In this case, we speak of the *rational restriction* of G to L, written $G|_L$.

More generally, one may wish to consider families of graphs generated by any kind of finitely represented relations over arbitrary countable domains, for instance arithmetical operations, relations between rectangular pictures or even between finite graphs.

2.1.2 External Presentations

We now enumerate several existing formalisms allowing to define families of infinite graphs in a finite manner without relying on the particular values (or 'names') of vertices. Rather, these methods directly describe the structure of graphs, either by transforming simpler graphs, by considering the sets of solutions of equations involving graph operators, or by evaluating sets of expressions over such operators. All these characterizations are referred to as *external*.

2.1.2.1 Graph Transformations

Many structural transformations over infinite graphs can be used to define families of infinite graphs which verify interesting properties. A fundamental idea is, starting from a single graph called *generator*, or from a family of generators, whose properties are already known, to obtain new graphs by applying transformations which preserve these properties. This section presents some of the most usual of these transformations.

We distinguish two main types of transformations: transformations increasing the number of vertices of a graph by either duplicating or unfolding it, and transformations defining new edge relations from the edge relations of a graph.

Copies, unfoldings and tree-graphs. We first enumerate a few external graph transformations able which produce a graph over an increased support. In the following, we fix G a Σ-labeled graph, and V the support of G.

K-copy. Let K be a finite alphabet disjoint from Σ, the K-copy operation applied to G simply duplicates $|K|$ times each vertex of G (while retaining all

edges between the original vertices) and adds an edge labeled by a different $k \in K$ between every original vertex of G and each of its $|K|$ copies. Formally, the K-copy G' of G is defined as the $(\Sigma \cup K)$-graph

$$\{u \xrightarrow{a} v \mid a \in \Sigma,\ s \xrightarrow[G]{a} t\} \cup \{u \xrightarrow{k} (u,k) \mid u \in V,\ k \in K\}.$$

Unfolding. Let v be a vertex of G, the unfolding of G from v is a tree whose branches represent all possible paths of origin v in G. Each node of this tree can be associated to a "history" of all the vertices and edges of G traversed by a given run. Formally, the unfolding of G from vertex v is the tree

$$\{wu \xrightarrow{a} wuau' \mid wu \in v(\Sigma V)^*,\ u \xrightarrow{a} u' \in G\}.$$

Nodes of this tree are identified with paths of origin v. These paths can be represented as non-empty sequences in the set $v(\Sigma V)^*$ of alternated vertices of G and edge labels in Σ, all starting with v.

Tree-graph. The tree-graph operation is a variant of a similar operation called *tree-iteration* [Wal02]. Consider the set V^+ of non-empty sequences of vertices of G. Let $\#$ be a new symbol, we define the tree-graph of G as the $(\Sigma \cup \{\#\})$-graph whose support is V^+ and whose set of edges is

$$\{wv \xrightarrow{\#} wvv \mid w \in V^+,\ v \in V\} \cup \{wu \xrightarrow{a} wv \mid u \xrightarrow{a} v \in G\}.$$

Monadic second-order interpretations. A standard operation in logics, called *interpretation*, consists in defining a new structure using the elements of an existing structure. We focus on interpretations of graphs inside graphs, and on the monadic fragment of second-order logic. Formally, a *monadic (second-order) interpretation* of Γ in Σ is a family $I = (\phi_a(x,y))_{a \in \Gamma}$ of MSO formulas over Σ-graphs with two free first-order variables. The interpretation of G by I is the Γ-graph

$$I(G) = \{u \xrightarrow{a} v \mid G \models \phi_a(u,v)\}.$$

Note that monadic interpretations do not allow to increase the number of vertices of a graph. To remedy this, one may consider compositions of transformations increasing the size of graphs with monadic interpretations. In the specific case of a K-copying followed by a MSO interpretation, one speaks of *monadic (second-order) transduction*. These very expressive transformations are presented in full detail in [Cou94] in the more general case of arbitrary structures.

Markings and restrictions. Even in the case where the naming of vertices in a graph is unknown, one may wish to select a certain set of vertices for subsequent use. This selection can of course only depend of the structure of the graph.

There are different ways to select subsets of vertices in a graph. First, given a vertex $r \in V$, one can consider the set $X_{(r,L)}$ of all vertices reachable in G by a path of origin r whose label belongs to some rational language $L \subseteq \Sigma^*$. Another way to select a set of vertices of G is, given a MSO-formula $\phi(x)$ with a single first-order free variable, to consider the set $X_\phi = \{v \in V \mid G \models \phi(v)\}$. In the former case, one speaks of rationally defined sets of vertices, and in the latter of MSO-definable sets.

Two common external operations over graphs exploiting these selections of vertices are markings and restrictions. Formally, the *marking* of a graph G with respect to a subset X of its vertices is the graph

$$M_\#(G, X) = \{u \xrightarrow{a} v \mid u \xrightarrow[G]{a} v\} \cup \{u \xrightarrow{\#} u \mid u \in X\}$$

obtained by adding in G a loop labeled by $\#$ to all the vertices in X, where $\# \notin \Sigma$ is a new symbol. We recall that the restriction of a graph G with respect to a set X is simply the graph $\{u \xrightarrow{a} v \mid u, v \in X,\ u \xrightarrow[G]{a} v\}$.

Markings and restrictions with respect to rationally defined and MSO-definable sets are respectively called rational and MSO-definable. Both operations are special cases of MSO-interpretation. A special case of MSO-restriction, called *restriction under reachability*, occurs when the selection formula matches the set of all vertices reachable from a given vertex.

Inverse substitutions. Another subclass of MSO-interpretations, called inverse rational or finite substitutions, defines the edges of a graph with respect to paths existing in the original graphs. A substitution is a mapping h from Γ to Σ^*, where Γ is an alphabet. It is called rational (resp. finite) if for all $a \in \Gamma$, $h(a)$ is a rational language (resp. a finite language). The inverse application of h to G is the Γ-graph

$$h^{-1}(G) = \{u \xrightarrow{a} v \mid \exists w \in h(a), u \xrightarrow[G]{w} v\}.$$

When $h(a) = \varepsilon^* a \varepsilon^*$ for all a, one rather speaks of $h^{-1}(G)$ as the *ε-closure* of G. This operation followed by a restriction to the set of external vertices yields an external characterization of the construction of the transition graph of a machine from its configuration graph.

Combining and repeating transformations. The particularity and main interest of all the above transformations is the fact that they preserve the decidability of the second-order monadic theory. More precisely, given a graph G with a decidable MSO theory, the image of G by a monadic second-order interpretation, by the tree-graph operation or by an unfolding, to cite but the most expressive, still has a decidable MSO theory.

This observation directly yields a method to characterize families of graphs with a decidable MSO theory. Let F be a family of graphs, one can characterize new families of graphs by application of some of the above transformations, or compositions or iterations thereof. The fact that all of these transformations preserve the decidability of the monadic second-order theory of graphs directly implies that the obtained families of graphs have a decidable MSO theory whenever F does.

2.1.2.2 Systems of Equations and Graph Grammars

All languages in the Chomsky hierarchy, as well as regular term languages for instance, can be defined using grammars. Grammars can be seen as systems of equations over appropriate operators, whose least solutions are the languages they define. One can define along the same idea grammars ranging over graphs, using appropriate sets of operators. The most widely used sets of graph operators are the *hyperedge replacement* (HR) and *vertex replacement* (VR) families of operators, both ranging over colored graphs (see for instance [Cou97] for a detailed presentation).

HR operators consist of finite colored graphs, the disjoint union operator, vertex recolorings and mergings of vertices of the same color. Systems of HR equations can be very naturally expressed using a family of graph grammars, which we call *hyperedge replacement grammars*. We only give an informal definition of these grammars.

Hyperedges can be pictured as arrows binding several vertices of a graph (in that case, a *hypergraph*), and formally defined as tuples $(a, v_1 \dots v_n)$, where $a \in \Sigma$ is a label and each v_i, $i \in [1, n]$ is a vertex, and n is the arity of the hyperedge. Sets of hyperedges in a graph are defined as labeled n-ary relations over vertices. A hyperedge-replacement grammar is characterized as a set of hyperedges labels called non-terminals, and a set of productions of the form $A \to H$, where A is a non-terminal and H is a colored hypergraph. If A is of arity n, we impose that exactly one vertex of H be colored by each $i \in [1, n]$.

Informally, a derivation step of such a grammar is defined by replacing each hyperedge labeled by a non-terminal A occurring in a hypergraph G by the corresponding right-hand side H in the grammar. Replacement of a hyperedge by a colored hypergraph is done by taking the disjoint union of G with H, merging the

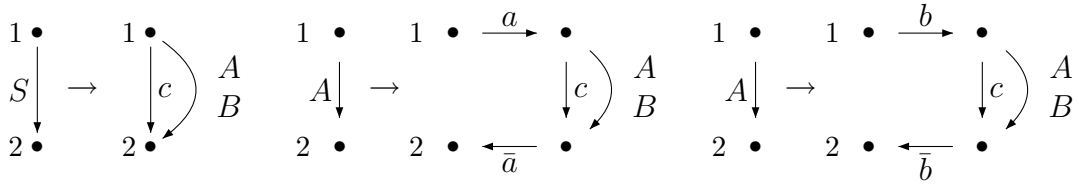


Figure 2.3: Hyperedge-replacement graph grammar.

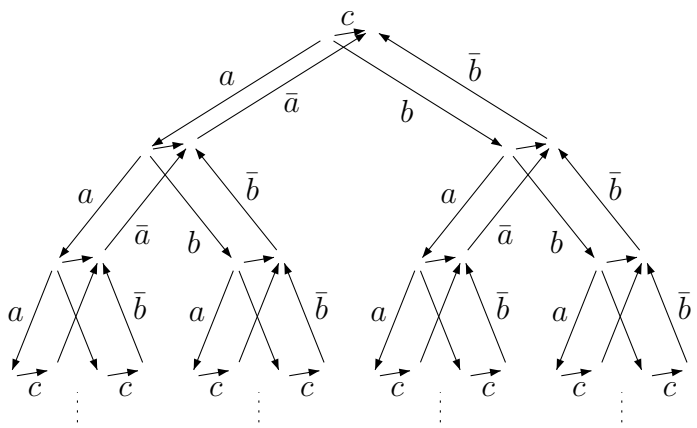


Figure 2.4: Infinite graph generated by the grammar of Figure 2.3.

---

i-th vertex of A with the i-colored vertex of H for each $i \in [1, n]$ and removing hyperedge A and all vertex colors from the obtained graph. Successive replacements of all hyperedges form a series of finite hypergraphs. The (infinite) graph generated by such a grammar is the limit of this series (which does not contain any non-terminal hyperedge). Figure 2.3 shows an example of a HR grammar. The graph it characterizes is represented in Figure 2.4.

VR operators allow the disjoint union of graphs, the addition of isolated colored vertices, the addition of edges between all pairs of vertices of given colors, and the recoloring of vertices as in HR sets of operations. Graphs generated by systems of equations over VR operators can also be characterized in terms of so-called vertex-replacement grammars, whose production rules specify possible replacements of non-terminal vertices with corresponding finite graphs. This involves precisely specifying the way former adjacent edges of the non-terminal vertex are reconnected to the graph it is being replaced with. We will not detail these notions further.

Other families extending VR and HR sets have been considered. In [Col04], the author uses several new operations to extend the VR set, in particular synchronous or asynchronous products of graphs, yielding respectively the VRS and VRA families, and provides equational characterizations of important families of graphs generalizing the VR-equational graphs (cf. Section 2.2).

2.1.2.3 Other External Characterizations

In the same way regular sets of words correspond to the sets of models of monadic second-order sentences, one can define sets of graphs as the models of closed formulas in any logical language over graphs. Given a logic L, one speaks of L-definable sets. Since variables in formulas do not refer to the naming of vertices, this is an external representation.

Finally, we mention characterizations of families of graphs by purely geometrical criteria. The most well-known instance of such a characterization concerns the family of pushdown graphs, which can be characterized as graphs of finite degree having finitely many decompositions by distance from any vertex (cf. Section 2.2.1.2).

2.2 Families of Infinite graphs

We saw in the previous section how to give finite presentations of interesting classes of infinite graphs. We also mentioned earlier why it is meaningful to investigate graphs up to isomorphism: we are interested in the structural properties of graphs, independently of the particular naming chosen for the vertices. We now illustrate these ideas on a selection of some of the most well-known families of infinite graphs.

2.2.1 Infinite Graphs with a Decidable MSO Theory

The monadic second-order logic is a very useful logic over graphs, balancing good expressiveness with a decidable satisfiability problem over several important logical structures.

2.2.1.1 Early Results

Büchi [Büc62] showed that the monadic second-order theory of the set of integers with the successor relation, $(\mathbb{N}, S)$, is decidable. This result can be interpreted as the decidability of MSO over the semi-infinite line, which is an infinite graph isomorphic to $(\mathbb{N}, S)$. The monadic second-order logic over this structure is sometimes called $S1S$, for (monadic) second-order logic of 1 successor relation.

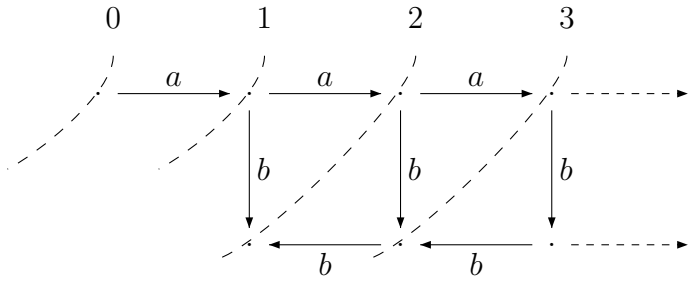


Figure 2.5: Distance decomposition of a pushdown graph from its root.

This result was later extended to another infinite graph by Rabin [Rab69], who showed that the monadic theory of the infinite complete binary tree Δ_2 is decidable. Similarly to the previous case, MSO over this structure is also called $S2S$, for obvious reasons. This result is referred to as the *Rabin tree theorem*. Many results concerning the decidability of MSO over more general families of infinite graphs are proved by a reduction to the logic $S2S$.

2.2.1.2 Pushdown Graphs

The real beginning of a study of infinite graphs for themselves was initiated by Muller and Schupp. In [MS85], they characterized the set of pushdown graphs, whose vertices are the configurations of a pushdown automata reachable from its initial configuration, and whose edges represent its transitions. According to our terminology, pushdown graphs are the restriction under reachability from the initial configuration of pushdown automata configuration graphs.

The main result concerning this family is that its graphs have a decidable MSO theory. This result was obtained by means of the following geometrical characterization. Let G be a graph and v_0 one of its vertices. We write $G_{\geq n}$ the restriction of G to its set of vertices whose distance to v_0 is greater than n (where the notion of distance we consider is the length of the shortest directed path between two vertices). Muller and Schupp proved that a graph is isomorphic to a pushdown graph if and only if the set of all connected components of the family $\{G_{v_0,n} \mid n \geq 0\}$ is finite up to isomorphism for all vertex v_0 (see Figure 2.5 for an illustration of this idea[1]). This result enabled them, using Rabin's tree theorem, to conclude that pushdown graphs have a decidable monadic theory.

Finally, it is quite straightforward to show that the languages of pushdown graphs between finite sets of vertices are the context-free languages.

[1]My thanks to Thomas Lavergne for lending me this picture.

2.2.1.3 Equational Graphs

Infinite graphs generated by hyperedge-replacement graph grammars are sometimes called regular or equational graphs. We will call them HR-equational for less ambiguity. Courcelle characterized and studied this family, and showed in particular that HR-equational graphs have a decidable monadic second-order theory ([Cou90]).

It was shown in [Cau92] that the family of rooted HR-equational graphs of finite degree coincides, up to isomorphism, with both the family of pushdown graphs and that of the restrictions under reachability of prefix rewriting graphs of finite word rewriting systems (or of the right closures of finite relations over words).

Without this restriction under reachability, one still gets an interesting equivalence between HR-equational graphs of finite degree, the rational restrictions of the right closures of finite word rewriting systems, the rational restrictions of configuration graphs of pushdown automata[2] ([Cau95]), and the rational right closures of finite relations ([Cau96]).

2.2.1.4 Prefix-Recognizable Graphs

Prefix-recognizable graphs were defined in [Cau96]. They are a good example of a family of infinite graphs which admits numerous different internal and external characterizations. This is summarized by the following equivalence theorem (only the most significant presentations are listed).

Theorem 2.7. *Let G be a labeled graph. The following statements are equivalent:*

1. *G is isomorphic to the rational right closure of a recognizable graph (or, equivalently, of the prefix rewriting graph of a recognizable word rewriting system) [Cau96],*
2. *G is isomorphic to the transition graph of a pushdown automaton [Sti98],*
3. *G is isomorphic to the Cayley-type graph of a prefix-overlapping rewriting system [CK02a],*
4. *G can be obtained by a rational marking of the infinite binary tree followed by an inverse rational substitution [Cau96],*
5. *G can be obtained by a monadic second-order interpretation of the infinite complete binary tree [Blu01],*
6. *G is VR-equational [Bar97].*

[2]Please note the nuance between Muller and Schupp's pushdown graphs, which are restricted to reachable vertices, and what we call here pushdown automata configuration graphs.

Each of these characterizations is credited to their respective authors. Several properties of this family directly ensue. Decidability of the MSO theory of prefix-recognizable graphs is a straightforward consequence of their external presentations by MSO-preserving transformations (items (4) and (5)). The fact that the traces of prefix-recognizable graphs are the context-free languages was already known from [Cau96], but can be seen as a consequence of (2). Finally, the closure of this family under rational restrictions and inverse rational substitutions is a consequence of (1).

Other consequences of these results concern subfamilies of prefix-recognizable graphs. For instance, the decidability of the MSO theory of VR-equational graphs can be directly deduced from (6). Also, it is pretty clear that HR-equational graphs are a subfamily of VR-equational ones. (As a matter of fact, it was shown in [Bar98] that HR-equational graphs precisely coincide with the family of VR-equational (or prefix-recognizable) graphs of bounded tree-width[3].) Since both HR-equational and pushdown graphs are strict sub-families of prefix-recognizable graphs ([Cau96]), this also provides another proof of the decidability of their respective monadic theories.

Also notice that, due to the fact that trace-equivalence is a much looser form of equivalence than isomorphism, there may be several strictly distinct families of graphs whose sets of traces form precisely the same family of languages. In the present case, since the traces of both families of pushdown graphs and prefix-recognizable graphs are the context-free languages, any intermediate family (for instance HR-equational graphs) has the same family of traces (see [Sti98] for more details).

2.2.1.5 The C. Hierarchy

In the spirit of prefix-recognizable graphs, a strict infinite hierarchy of families of graphs obtained through iteration of graph transformations was studied in [Cau02]. Level 0 of the hierarchy consists of the set of finite graphs. Graphs at level $n > 0$ are defined as the unfoldings of graphs of level $n - 1$, followed by an inverse rational substitution. For later reference[4], we will call this hierarchy the 'C. hierarchy'. A direct consequence of this definition is that all the graphs of the C. hierarchy have a decidable monadic theory. Further studies of this hierarchy and comparisons with other families of graphs lead to the following equivalence result.

Theorem 2.8. *Let G be a labeled graph. The following statements are equivalent:*

[3]The tree-width of a graph is a measure of its similarity to a tree. Its definition is not trivial and not relevant to this work.

[4](and for the sake of modesty and discretion)

1. *G belongs to level n of the C. hierarchy (i.e. is defined as the application to a finite graph of n successive unfoldings composed with inverse rational mappings),*
2. *G can be obtained by the application to a finite graph of n successive tree-graph operations composed with MSO transductions ([CW03]),*
3. *G is isomorphic to the transition graph of a level n higher-order pushdown automaton ([CW03]),*
4. *G is isomorphic to the graph of a 'higher-order prefix-recognizable' relation ([Car05]).*

Furthermore, Caucal shows in [Cau02] that the trees obtained by unfolding the graphs of the considered hierarchy are isomorphic to both the solutions of *safe* higher-order schemes [KNU02], and to a hierarchy of terms defined by a form of iterated first-order substitution [CK02b].

As the transition graphs of higher-order pushdown automata, the traces of the graphs of this hierarchy quite naturally coincide with the OI hierarchy of languages (cf. Section 1.3.4). Recall that these languages are a subfamily of context-sensitive languages.

2.2.2 Rational and Automatic Graphs

The family of rational graphs owes its name to the fact that their sets of edges are given by rational relations on words, i.e. relations recognized by word transducers. In this section, we present the general family of rational graphs and three of its sub-families, namely the synchronized, synchronous and sequential synchronous rational graphs.

Rational graphs are simply defined as the computation graphs of finite families of word transducers (cf. Section 2.1.1.1). Hence, any Σ-labeled rational graph G is characterized by a family $(T_a)_{a\in\Sigma}$ of word transducers over some alphabet Γ. We recall that, by definition of computation graphs, there is an a-edge in G between vertices u and $v \in \Gamma^*$ if and only if $(u, v) \in T_a$.

In [Mor00], alternative characterizations of this family are given. They are shown to coincide with the rational restrictions of a subfamily of inverse linear[5] substitutions of the complete binary tree.

Figure 2.6 shows an example of rational graph, the infinite *grid*, with the rational transducers which define its edges. Note that it follows from Lemma 1.29 that the support of a rational graph is a rational subset of Γ^*.

The rational graphs with synchronized transducers were already defined by Blumensath and Grädel in [BG00] under the name *automatic* graphs and by

[5] A language is linear if it is accepted by a context-free grammar in which all right-hand sides of rules contain at most one non-terminal.

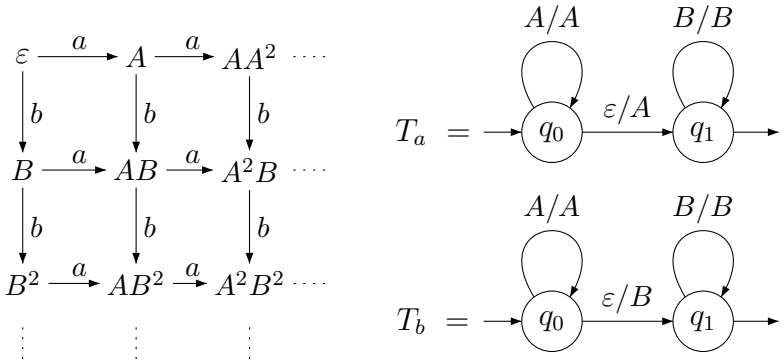


Figure 2.6: The grid and its associated transducers.

Rispal in [Ris02] under the name *synchronized* rational graphs. By a slight abuse of language, we also refer to rational graphs with synchronous transducers as *synchronous* graphs, and with synchronous and sequential transducers as *sequential synchronous* graphs.

It follows from the definitions (see Section 1.4.2) that sequential synchronous, synchronous, synchronized and rational graphs form an increasing hierarchy. This hierarchy is strict (up to isomorphism): first, sequential synchronous graphs are deterministic graphs whereas synchronous graphs can be non-deterministic. Second, synchronous graphs have a finite degree whereas synchronized graphs can have an infinite degree. Finally, to separate synchronized graphs from rational graphs, we can use the following properties on the growth rate of the out-degree in the case of graphs of finite out-degree.

Proposition 2.9. *[Mor01] For any rational graph G of finite out-degree and any vertex x, there exists $c \in \mathbb{N}$, such that the out-degree of vertices at distance n of x is at most c^{c^n}.*

This upper bound can be reached: consider the unlabeled rational graph $H = \{T\}$ where T is the transducer over $\Gamma = \{A, B\}$ with one state q_0 which is both initial and final and a transition $q_0 \xrightarrow{X/YZ} q_0$ for all X, Y and $Z \in \Gamma$. It has an out-degree of $2^{2^{n+1}}$ at distance n of A. In the case of synchronized graphs of finite out-degree, the bound on the out-degree is simply exponential.

Proposition 2.10. *[Ris02] For any synchronized graph G of finite out-degree and any vertex x, there exists $c \in \mathbb{N}$, such that the out-degree of vertices at distance n of x is at most c^n.*

It follows that the graph H above is rational but not synchronized. Hence, the synchronized graphs form a strict sub-family of rational graphs.

Rational graphs have an undecidable MSO theory, even in the synchronous case. Synchronized graphs have a decidable first-order theory, but there exist general rational graphs with an undecidable first-order theory.

One very important property of rational graphs is that their languages between rational (or finite) sets of initial and final vertices are the context-sensitive languages [MS01]. This result was later extended to synchronized graphs [Ris02], and event to synchronous ones when only rational sets of initial and final vertices are considered. In Section 4.2, we give simple proofs of these properties, and in Section 4.4 we provide a comparison of rational graphs with another class of infinite graphs whose languages are the context-sensitive languages.

2.2.3 Turing Graphs

Turing graphs, sometimes also called computable graphs, are the transition graphs of Turing machines. They were studied in [Cau03]. In this paper, the author shows that Turing graphs can be characterized in several different ways: they coincide with the computation graphs of Turing machines seen as transducers, with the inverse rational substitutions of rational graphs, with the rational restrictions of images of the infinite binary tree by any sufficiently general family of inverse substitutions (where 'sufficiently general' is given a precise meaning in terms of a subfamily of context-free languages), and with the Cayley-type graphs of unrestricted rewriting systems.

2.2.4 Graphs over Terms

Since they admit internal presentations as rewriting systems, most previous families of graphs can be naturally extended to graphs whose vertices are terms. Pushdown graphs are extended to terms using the restrictions under reachability of ground rewriting graphs. By [DHLT90], these graphs have a decidable first-order theory with reachability. They were also studied by Löding in [Löd02]. Replacing the restriction under reachability by a restriction to a deterministic top-down set of terms yields an extension of the prefix rewriting graphs of finite systems, and prefix-recognizable graphs can be extended the same way by considering recognizable sets of rewriting rules [Col02]. Finally, term-automatic graphs were considered in [BG00] using a notion of synchronized term relations equivalent to that of Section 1.4.3.3. All these families of graphs were characterized in terms of systems of equations in [Col04].

2.3 Infinite Graphs and Verification

Nowaday's main challenge in the field of automatic verification is to extend existing methods based on finite automata (cf. Section 1.6), whose success is now widely recognized, to the case of infinite-state systems. This need is due to the fact that finite-state abstractions of systems do not in general allow the verification of very accurate or subtle properties of systems. Several aspects of systems are sources of infiniteness, which either relate to infinite data domains, or to an infinite control mechanism.

2.3.1 Symbolic Model Checking

The model checking problem consists in determining whether a given system verifies a certain property expressed in a temporal or classical logic. Among the simplest properties of interest lie *safety properties*, which concern the reachability relation in a system. A common instance of such a property can be formulated as the question *'Is it possible to reach a configuration in B from any configuration in A?'* where A and B are sets of configurations of the system under consideration.

To answer this type of questions over infinite systems, a primary requirement is to be able to represent potentially infinite sets of configurations in a finite manner, and compute their images and inverse images by the system's transition relation, or even better by the transitive closure of this relation. The chosen representation also has to be closed under basic set operations such as union and intersection. This framework is usually called symbolic verification.

A very common choice for the symbolic representation of sets of configurations encoded as words is to use regular languages, which enjoy many interesting closure and algorithmic properties. In that case, one more specifically speaks of *regular model checking*. Recently, many works have shown that finite-state automata are suitable generic representation structures for sets of configurations, which allow to uniformly handle a wide variety of systems including pushdown systems, FIFO-channel systems, parameterized networks of processes, counter systems, etc. [BEM97, BGWW97, KMM+97, ABJ98, WB98, BJNT00, Bou01, HJJ+95]. In all cases, a key property is the *preservation of regularity* by the system's transition relation (or its transitive closure): we say a relation preserves regularity (by inverse) if the image (resp. pre-image) of any regular set is regular. Of course, all of these ideas can be generalized from words to terms, since regular sets of terms share most of the good properties of regular sets of words. Extensions of existing techniques to terms can be found in [AJMD02, BT02] for instance.

Many of the models expressive enough to represent interesting classes of programs and systems immediately yield undecidable verification problems for all but the simplest properties. For instance, it is very straightforward to encode the

computations of a Turing machine as the transitions of a linear parameterized network, which makes the reachability of any given configuration in such a model undecidable.

There are two main approaches to deal with this negative observation. The first one is to consider approximated algorithms or semi-decision procedures to give partial answers to otherwise unsolvable problems. The second one, which is more relevant to our approach, is to reduce the expressiveness of the considered models, and characterize classes of systems over which the above-mentioned verification problems are decidable. In the remaining of this section, we mention two active topics in the community of infinite-state verification whose formulation admits a direct rephrasing in terms of infinite graphs.

2.3.2 Verification of Recursive Programs

The most obvious class of programs in which, even after the standard finite-state abstraction of variables, an infinite number of configurations has to be considered is programs containing mutually recursive procedures. Such programs are traditionally modeled using a natural transcription in terms of pushdown systems (i.e. pushdown automata without input). Verification of properties in this setting can then directly be seen as a model checking problem over configuration or transition graphs of pushdown automata. Pushdown systems and their related decision and algorithmic analysis problems (reachability analysis, model checking, games solving and control synthesis, etc.) have been widely investigated in the last few years [Cau92, BCS96, Wal96, BEM97, EHRS00, Cac02, AEM04].

New interesting results concerning higher-order pushdown automata and the decidability of the MSO theory of their transition graphs have interesting consequences in terms of program verification. Indeed, in the same way pushdown automata can model programs with recursive procedure calls, higher-order pushdown automata can be used as models for *higher-order functional programs*. Hence, a natural question is to investigate possible extensions of existing techniques from pushdown systems to the higher-order case. Chapter 5 presents an attempt at proposing a symbolic reachability analysis algorithm adapted from [BEM97] for a subclass of higher-order pushdown systems.

2.3.3 Parameterized and Dynamic Systems

Another class of models, referred to as parameterized systems, aims at verifying a program or system independently of the value given to a certain parameter, or even to synthesize values of the parameter for which a given property hold or doesn't hold. For all instantiations of the parameter, the obtained model is finite, but one has to check an infinite number of arbitrarily large systems in order to

check the property.

An instance of such problems is to consider fixed networks of communicating processes sending and receiving messages through unbounded communication channels. The topology of the considered networks may be linear, tree-like, or even graph-like in the worst case. The parameter in question is the initial topology of the network and the state of each of its processes. Processes may evolve by changing their state while synchronizing with their direct neighbors in the network. Properties one wants to verify may concern the liveness of the network (i.e. the absence of deadlocks), the reachability of a global terminal state, or more general properties expressed in any given logic. Linear and tree-like networks are very naturally represented as words and terms, where each symbol in the alphabet corresponds to a finite abstraction of the current state of the corresponding process, and graph-like networks by colored graphs. Transitions of the system can be modeled by length-preserving rewriting rules (structure-preserving in the case of terms and graphs).

More difficult problems arise when modeling networks of processes in which new processes can be dynamically allocated and deallocated, communication channels opened and closed, and so on. The transitions of such systems are modeled using general rewriting rules. Chapter 3 investigates classes of term rewriting systems whose derivation relation can be computed and finitely represented, which gives rise to potential applications in this framework.

Chapter 3

Term-Rewriting Systems with a Rational Derivation

One of the arguments of Section 2.1.1.2 is that rewriting systems provide a very natural framework for the internal characterization of families of infinite graphs. Rewriting systems whose expressiveness match any common type of language acceptors are very easy to define while providing a better flexibility, in particular to model various classes of systems for verification purposes, like parameterized networks of communicating processes (cf. Section 2.3). In that setting, an important concern is to be able to compute the derivation relation of a given rewriting system, or in some cases the image of a regular set by this relation.

This chapter's concern is to extend successful approaches in the case of word rewriting systems to terms, focusing on the exact computation of derivation relations. In [Cau00], a classification of word rewriting systems according to a precise syntactic condition on the set of rewriting rules is led, with the objective to identify classes whose derivation relations are rational (i.e. accepted by finite transducers). As a result, the classes of left, right, prefix and suffix word rewriting systems were defined, and it was shown that the derivation relations of these systems are rational and effectively computable. Better still, in the case of left and right systems they precisely coincide with the class of rational relations: left and right rewriting rules can be seen as a relaxed syntactical form of transducer transition rules.

The classification criterion used in [Cau00] can be readily adapted to terms, as shown in Section 3.2 below, and natural extensions of prefix, suffix, left and right systems defined. However, to properly extend the previous result to these classes, we need a suitable class of binary relations over terms having a finite presentation, and for which membership is decidable. It turns out that ordinary term transducers are not expressive enough to satisfactorily capture the derivations of all the classes we consider. For this reason, we chose to use the class of rational term

relations defined in [Rao97], which is recalled in Section 3.1. Finally, a detailed exposition of the obtained results is given in the remaining sections. It turns out that one of the considered classes of systems has non-recursive derivations, while the three others' derivations are rational in the sense of [Rao97].

3.1 Rational Tree Relations

As discussed in Section 1.4.3.4, several attempts at defining extensions of rational relations from words to terms have been made, and many different notions of binary relations over terms exist. In that section, we recalled a few characteristics of rational word relations, and established them as a kind of checklist to determine how close a particular extension is to the word case.

In [Rao97], a notion of rationality for tuples of trees according to the union, substitution and iterated substitution operations is proposed. It can also be seen as a definition for rational binary relations over tuples of terms, and thus as a special case, binary relations over terms. This definition is justified by its similarity to the word case on several aspects: it coincides with rational word relations when restricted to words (seen as terms over symbols of arity at most 1), it has a decidable membership test, it is closed under union and converse, and finally it can be characterized using at least two natural formalisms, namely rational expressions and grammars over tuples of trees. The main difference between these relations and rational relations on words is that they are not closed under composition.

Before giving the definition of rational sets of tuples of terms, we introduce some additional notation.

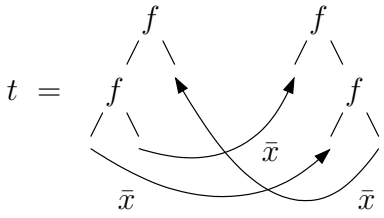
3.1.1 Multivariables

We generalize the notion of term variable in order to be able to bind together several positions in a term word. Intuitively, we associate arities to variables, with the idea that a variable of arity n (called to avoid confusion a *multivariable*) is related to a n-tuple of positions in a term or term word. For easier writing, we decompose each multivariable of arity n into a sequence of n unique single variables.

Formally, let F be a ranked alphabet and X a set of variables. We call *multivariable* a word $\bar{x}$ in X^* containing at most one occurrence of any variable. For instance, $\bar{x} = x_1x_2x_3$ is a multivariable of length 3. We want to be able to consider term words containing more than one occurrence (or *instance*) of the same multivariable. To avoid confusion and be able to assign each variable to the appropriate multivariable, we need to explicitly distinguish each instance

of a given multivariable. Let $\bar{x}$ be a multivariable, the set of instances of $\bar{x}$ is written $\{\bar{x}^i \mid i \in \mathbb{N}\}$. When picturing term words as forests, we represent each multivariable of arity n as an arrow binding together n leaves of the term word.

Example 3.1. Let $\bar{x} = x_1x_2$ be a multivariable of length 2. The linear term word $t = f(f(x_1^1, x_1^3), x_2^2)\ f(x_2^3, f(x_2^1, x_1^2))$ contains three instances of $\bar{x}$, respectively written $x_1^1x_2^1$, $x_1^2x_2^2$ and $x_1^3x_2^3$. We can represent t graphically as follows:



Let $\overline{X} \subseteq X^*$ be a set of multivariables. We assume that, for all $\bar{x} = x_1 \dots x_n, \bar{y} = y_1 \dots y_m \in \overline{X}$, all variables x_i and y_j of $\bar{x}$ and $\bar{y}$ are distinct. In this chapter, we will be considering only linear term words over F whose variables all belong to one, and only one, complete instance of a multivariable (where a multivariable instance is complete if none of its variables is missing)[1]. The set of such term words is denoted by $T^*(F, \overline{X})$. When only one instance $\bar{x}^1$ of $\bar{x}$ occurs in a term word, we simply denote it by $\bar{x}$. Two term words are considered as equal if they only differ by the numbering of their multivariable instances.

Example 3.2. Let $\bar{x} = x_1x_2x_3$ be a multivariable of length 3. The term fx_1x_3 does not belong to $T(F, \overline{X})$ because it contains an incomplete instance of $\bar{x}$.

The substitution operation is also specialized to this setting. We abbreviate as $\{\bar{x}^i \mapsto s\}$ the substitution $\{\bar{x}^i_j \mapsto s_j \mid j \in [1, n]\}$, where $\bar{x}^i = \bar{x}^i_1 \dots \bar{x}^i_n$ is an instance of multivariable $\bar{x}$ and $s = s_1 \dots s_n$ is a term word. Since we only consider linear term words, when performing a substitution $t\{\bar{x}^i \mapsto s\}$ we also suppose that the numbering of any instance of a multivariable in s is chosen greater than that of other instances of the same multivariable already occurring in t.

Example 3.3. Let $\bar{x} = x_1x_2$ be a multivariable, $\sigma = \{\bar{x}^1 \mapsto ab,\ \bar{x}^2 \mapsto cd\}$ a substitution and $t = f(f(x_1^1, x_2^1), x_2^2)\ x_1^2$ a term with two instances of $\bar{x}$, we have $t\sigma = f(f(a, b), d)\ c$.

[1] In this setting, the total number of distinct variables possibly appearing in a term is not bounded, but is finite.

3.1.2 Rational Sets of Tuples of Terms

We now state the definition of a product operation over term words, from which a notion of rationality over this set is defined. This product is an extension of the usual substitution operation over linear terms. Let t be a term word in $T^*(F, \overline{X})$, $\bar{x}$ a multivariable of n variables with $k \geq 0$ instances $\bar{x}^1 \dots \bar{x}^k$ in t (i.e. a total of $n * k$ variables), and M a set of term words of length n. We define $t \cdot_{\bar{x}} M$ as the set of tuples of terms obtained by replacing each instance of $\bar{x}$ in t with a (possibly different) term word in M. Formally,

$$t \cdot_{\bar{x}} M = \left\{ t\{\bar{x}^1 \mapsto s_1, \dots \bar{x}^k \mapsto s_k\} \mid \forall i \in [1,k], s_i \in M \right\}$$

where k is the number of occurrences of $\bar{x}$ in t. This operation is extended to sets by additive extension in the usual way: $L \cdot_{\bar{x}} M = \{t \cdot_{\bar{x}} M \mid t \in L\}$. We insist on the fact that the length of multivariable $\bar{x}$ and of all term words in M must be *the same* for this product to be defined.

If L is a set of term words of length n and $\bar{x}$ is still a multivariable of length n, we define $L^{0_{\bar{x}}} = \{\bar{x}\}$ and $L^{n_{\bar{x}}} = \{\bar{x}\} \cup L \cdot_{\bar{x}} L^{n-1_{\bar{x}}}$. The star of L over multivariable x is then as usual $L^{*_{\bar{x}}} = \bigcup_{n \geq 0} L^{n_{\bar{x}}}$. We can now define the notion of rationality associated to this product operation.

Remark 3.4. It is very important to understand that, due to the definition of concatenation, different instances of the same multivariable may be substituted with different term words in a product. For instance, let $F = \{f^{(2)}, a^{(0)}\}$, the rational expression $(fx^1x^2)^{*_x} \cdot_x a$ generates the whole set of terms $T(F)$ and not, as one might first think, the set of balanced terms over F.

Definition 3.5 ([Rao97])**.** The family Rat_n of rational sets of n-tuples of trees is the smallest family containing the finite sets of tuples and closed under the following operations:

1. $L \in Rat_n \ \wedge \ M \in Rat_n \ \Rightarrow \ L \cup M \in Rat_n$
2. $L \in Rat_n \ \wedge \ \bar{x} \in X^m \ \wedge \ M \in Rat_m \ \Rightarrow \ L \cdot_{\bar{x}} M \in Rat_n$
3. $L \in Rat_n \ \wedge \ \bar{x} \in X^n \ \Rightarrow \ L^{*_{\bar{x}}} \in Rat_n$

The family Rat of rational sets of tuples is the union of all Rat_n, for $n \geq 0$.

One should note that this notion of rationality differs from the one defined in [LW01], for example, as the concatenation (or 'series') product is not directly taken into account, and substitution is done simultaneously on several variables. From this definition arises a straightforward notion of *rational expression*, which extends the usual notion on words. It should be noted that even Rat_1 does not coincide with the set of regular term languages, as illustrated by the following example.

Example 3.6. The rational expression $fxy \cdot_{xy} (gxgy)^{*_{xy}} aa$ over alphabet $F = \{f^{(2)}, g^{(1)}, a^{(0)}\}$ represents the language $\{fg^n ag^n a \mid n \geq 0\} \in Rat_1$, which is not a regular term language.

3.1.3 Grammars for Tuples of Terms

Let us now recall the grammars used in [Rao97] to generate sets of term words.

Definition 3.7 (Multivariable grammar)**.** A *multivariable grammar* (or simply here a grammar) is a tuple $G = (F, N, S, P)$, where F is a ranked alphabet, N is a set of distinct multivariables called non-terminals, $S \in N$ is the initial non-terminal or axiom, and P is a set of productions. By analogy with word grammars, we write non-terminals using capital letters. Productions are of the form (A, t), also written $A \to t$ with $A = A_1 \dots A_n \in N$ and $t = t_1 \dots t_n \in T(F, N)^n$. We call A the left-hand side and t the right-hand side of the rule.

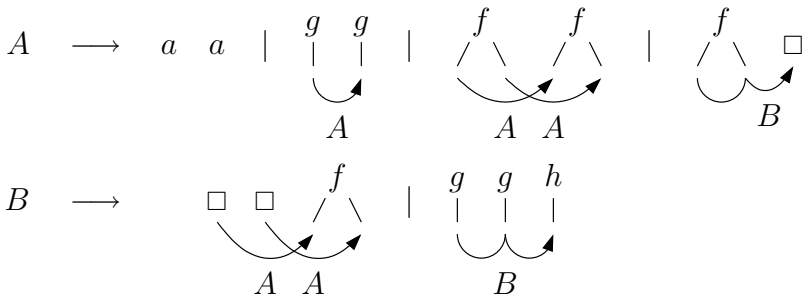
A derivation step of G from term word s is defined as the substitution of *one instance* of a non-terminal A by a corresponding right-hand side t in s. Formally, we write $s \underset{G}{\longrightarrow} s'$ if $s' = s\{A^i \mapsto t\}$ where $(A, t) \in P$ and A^i is an instance of A in s. As usual, the reflexive and transitive closure of $\underset{G}{\longrightarrow}$ is written $\underset{G}{\xrightarrow{*}}$. The language generated by G from axiom S is the set of ground term words $L(G) = \{w \in T(F)^{|S|} \mid S \underset{G}{\xrightarrow{*}} w\}$. Of course, a grammar whose axiom S is of size n only generates term words of length n.

In [Rao97], multivariable grammars are pictured as a specific kind of hyperedge replacement grammars (cf. Section 2.1.2.2). Let (A, t) be a production rule, we see A as a hyperedge of arity $|A|$ and t as an ordered forest in which all variables belonging to the same instance of a given non-terminal are represented as hyperedges labeled by the non-terminal in question. Of course, due to the definition of multivariable grammars, the obtained graph grammars have a very specific shape, to which they owe the name 'leaf-linked forests'. Let us illustrate this on an example.

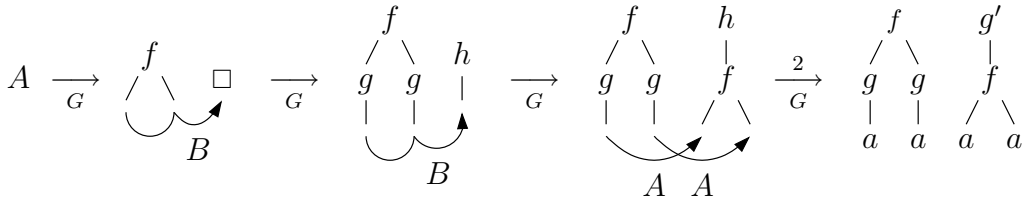
Example 3.8. Let $A = A_1A_2$ and $B = B_1B_2B_3$ be two non-terminals of respective arities 2 and 3. Grammar G_1 with rules

$$\begin{aligned} A &\longrightarrow a\ a \mid gA_1\ gA_2 \mid fA_1^1A_1^2\ fA_2^1A_2^2 \mid fB_1B_2\ B_3 \\ B &\longrightarrow A_1^1\ A_1^2\ fA_2^1A_2^2 \mid gB_1\ gB_2\ hB_3 \end{aligned}$$

can be represented as the hyperedge replacement grammar



An example of a production sequence of G_1 is:



This is just another way to represent multivariable grammars, and the definition of grammar derivation does not change and simply follows that of multivariable grammar derivation. However, one can observe that it is consistent with the usual definitions for hyper-edge replacement grammars. As expected, multivariable grammars generate precisely the rational sets of term words.

Theorem 3.9 ([Rao97])**.** *A set of term words is rational if and only if it is the language generated by a multivariable grammar.*

Example 3.10. The set of term words generated by G_1 from A can be represented by the rational expression

$$\big(\ (fxyz)\cdot_{xyz}(gx\ gy\ hz)^{*_{xyz}}(u^1\ u^2\ fv^1v^2)+(fu^1u^2\ fv^1v^2)+(gu\ gv)+(a\ a)\ \big)^{*_{uv}}.$$

3.1.4 Sets of Term Words as Binary Relations

A rational set of term words of length n can also be seen as a binary relation in $T(F)^p \times T(F)^q$, where $p+q=n$. In this case, given a non-terminal A, we define the first and second projections $\pi_1(A)$ and $\pi_2(A)$ by the set of variables of A referring to the first (resp. second) projection of the relation. A similar notation is used for right-hand sides of grammar productions as well. For clarity, we write a production (A,t) as $(A,\pi_1(t)\times\pi_2(t))$. Without loss of generality, we always consider that non-terminals are defined in such a way that $A=\pi_1(A)\pi_2(A)$ and $t=\pi_1(t)\pi_2(t)$. For example, if the axiom of a $T(F)^p\times T(F)^q$ relation is $A=A_1\dots A_n$, we can have $\pi_1(A)=A_1\dots A_p$ and $\pi_2(A)=A_{p+1}\dots A_{p+q=n}$.

Example 3.11. Grammar G_1 from Ex. 3.8 generates, from non-terminal A, a language $L(G_1, A) \in Rat_2$, which can be seen as a $T(F) \times T(F)$ relation. In this case, its rules can be written

$$A \longrightarrow a \times a \mid gA_1 \times gA_2 \mid fA_1^1A_1^2 \times fA_2^1A_2^2 \mid fB_1B_2 \times B_3$$
$$B \longrightarrow A_1^1 A_1^2 \times fA_2^1A_2^2 \mid gB_1\, gB_2 \times hB_3$$

This class of relations is closed under intersection with recognizable relations and under converse but not under composition, and does not preserve regularity in general, i.e. the image and pre-image of a regular set of terms by such a relation may not be regular. However, [Rao97] isolates a subclass of rational term relations which is closed under composition and preserves regularity, called rational *transductions*.

Definition 3.12 (Rational term transduction)**.** A rational relation is a transduction if it is generated by a grammar G for which there exists an integer k such that, for all derivation $A \xrightarrow[G]{\geq k} t$, all multivariable instances occurring in term word t have variables in at most two terms, one in each projection of the relation.

3.2 Classification of Term Rewriting Systems

In the case of words, several natural classes of rewriting systems can be distinguished by the way their rules are allowed to overlap. In [Cau00], the composition $\underset{R}{\longrightarrow} \circ \underset{R}{\longrightarrow}$ of two rewriting steps is considered, and all the different possibilities of overlapping between the right-hand side of the first rewriting rule, and the left-hand side of the second one are examined. Two words u and v overlap if either a suffix of one is the prefix of the other, or one is a factor of the other. By discarding systems where unwanted overlappings occur, one obtains four general families of systems whose derivation is proved rational, the families of *left*, *right*, *prefix* and *suffix* word rewriting systems. Moreover, it is proved that all other classes contain at least one system which has a non-rational derivation.

Since terms generalize words, and non-rational relations on words can not be generated by multivariable grammars either, we only need to study the extension of these four families of systems to terms: the classes of *bottom-up* and *top-down* systems, which respectively correspond to left and right systems, and the families of *prefix* and *suffix* systems.

The notion of overlapping we consider is the same as for words, with a slight nuance concerning variables: two terms overlap if one is a sub-term of the other, or a prefix of one is a suffix of the other *up to variable renaming*. Formally, let R be a rewriting system over some alphabet F, an *overlapping* between a right-hand

side r and a left-hand side l of rules of R is a 5-tuple (l, r, c, o, σ) with $c \in C(F)$, $o \in T(F, X)$ and $\sigma \in S(F, X)$ such that either:

- $c[o] = r$, $o\sigma = l$ and o is non-trivial (top-down overlapping),
- $c[o] = l$, $o\sigma = r$ and o is non-trivial (bottom-up overlapping),
- $o = l$, $c[o\sigma] = r$ and neither c nor σ is trivial (strict domain-internal overlapping),
- $o = r$, $c[o\sigma] = l$ ant neither c nor σ is trivial (strict image-internal overlapping).

These different kinds of overlappings include the 7 types of overlapping originally considered in the case of words (cf. [Cau00] for further details). By non-trivial substitution, we explicitly refer here to a substitution whose application to the concerned term adds at least one functional symbol (and excludes in particular mere variable renamings). Note that by hypothesis, we only consider rewriting systems whose set of rules is closed under variable renaming.

Given a system R over alphabet F, we consider the set O_R of all overlappings between right- and left-hand sides of R. Note that there may exist several different overlappings for given right- and left-hand sides of R. After discarding all unfavorable cases, we retain the following four classes of systems. A term rewriting system R is said:

- *top-down* if any overlapping in O_R is top-down (and only top-down),
- *bottom-up* if any overlapping in O_R is bottom-up (and only bottom-up),
- *prefix* if any overlapping (l, r, c, o, σ) in O_R is such that c is trivial,
- *suffix* if any overlapping (l, r, c, o, σ) in O_R is such that σ is trivial.

Note that the inverse of a top-down system is bottom-up (and conversely), the inverse of a prefix system is prefix and the inverse of a suffix system is suffix. Figure 3.1 illustrates these four types of systems. Prefix and suffix systems respectively generalize root and ground rewriting systems.

The next section establishes the rationality of derivations of top-down (and thus also bottom-up) systems. Contrary to the case of words, where prefix and suffix systems are dual and share the same properties, the situation is different in the case of terms. Section 3.2.2 presents a short proof of the well-known fact that these systems are Turing-powerful. Section 3.2.3, however, proves that any suffix system has a rational derivation.

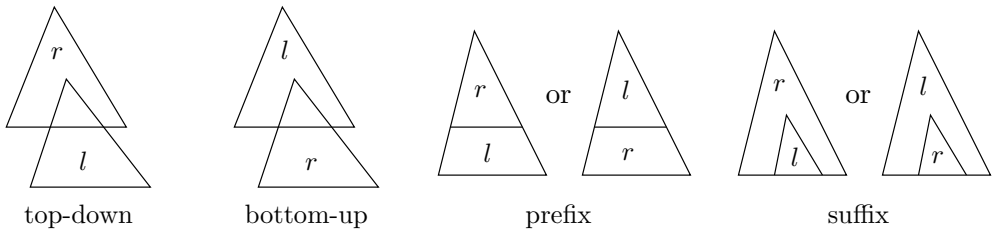


Figure 3.1: Types of overlappings in top-down, bottom-up, prefix and suffix systems.

3.2.1 Bottom-Up and Top-Down Systems

This section focuses on the study of top-down and bottom-up term rewriting systems and their derivations. All the proofs we provide consider top-down systems, but all results extend to the bottom-up case by duality (see Corollary 3.18).

For any finite linear top-down system, we show that a grammar generating its derivation relation can be built, which implies that this relation is rational. Furthermore, from the shape of the grammar, we observe that the derivation of such a system preserves the regularity of term languages. Dual results can be obtained for bottom-up systems: the derivation of a linear bottom-up system is rational, and the inverse image of a regular term language is still regular.

3.2.1.1 Monotonic Rewriting

Let us first observe that top-down systems enjoy a kind of *monotonicity* feature. Any rewriting sequence of such systems is equivalent to a sequence where the successive rewriting steps occur at non-decreasing positions in the input term. We call this *top-down rewriting*. Let R be a term rewriting system, we define its top-down rewriting

$$\xhookrightarrow[R]{*} = \bigcup_{n\geq 0} \xhookrightarrow[R]{n} \text{ with } \begin{cases} \xhookrightarrow[R]{0} &= Id_{T(F)} \\ \xhookrightarrow[R]{n} &= \bigcup_{p_1,\ldots,p_n} \xrightarrow[u_1,v_1]{}{}^{p_1} \circ \ldots \circ \xrightarrow[u_n,v_n]{}{}^{p_n} \end{cases}$$

such that the rewriting positions do not decrease along indexes ($\forall i, j,\ i < j \ \Rightarrow \neg(p_j < p_i)$), and if two successive positions are equal then the second rewriting should not have a trivial left-hand side ($(p_i = p_{i-1}) \ \Rightarrow \ (u_i \notin X)$). This last condition means that, for instance, the sequence

$$c[l\sigma] \xrightarrow[l,r]{} c[r\sigma] \xrightarrow[x,r']{} c[r'\{x \mapsto r\sigma\}]$$

is not top-down, because the second rule produces its right-hand side 'higher' than the first one. The rewriting steps should be swapped to obtain the top-down sequence

$$c[l\sigma] \underset{x,r'}{\longrightarrow} c[r'\{x \mapsto l\sigma\}] \underset{l,r}{\longrightarrow}^{pos(x,c[r'])} c[r'\{x \mapsto r\sigma\}].$$

The next lemma expresses the fact that, given any rewriting sequence of a top-down system, rewriting steps can always be ordered into an equivalent top-down sequence.

Lemma 3.13. *The relations of derivation and top-down derivation of any top-down term rewriting system R coincide: $\overset{*}{\underset{R}{\longrightarrow}} = \overset{*}{\underset{R}{\hookrightarrow}}$.*

Proof. By definition, $\overset{*}{\underset{R}{\hookrightarrow}} \subseteq \overset{*}{\underset{R}{\longrightarrow}}$. It remains to prove the converse, by using the same technique as in [Cau00]. Without losing generality, we assume that $R \cap (X \times X) = \emptyset$. Then, we can sort any derivation into a *top-down* derivation, as defined above, by applying an alternative version of the bubble sort algorithm in which removal and addition of elements are allowed. To do so, we use the following inclusions:

$$\underset{u,v}{\longrightarrow}^{\geqslant p} \circ \underset{x,v'}{\longrightarrow}^{p} \subseteq \underset{x,v'}{\longrightarrow}^{p} \circ \underset{u,v}{\longrightarrow}^{2^{>p}} \tag{3.1}$$

$$\underset{u,v}{\longrightarrow}^{>p} \circ \underset{u',v'}{\longrightarrow}^{p} \subseteq \underset{u',v'}{\longrightarrow}^{p} \circ \underset{u,v}{\longrightarrow}^{2^{\geqslant p}} \quad \text{with } u, u' \notin X \tag{3.2}$$

$$\underset{u,v}{\longrightarrow}^{>p} \circ \underset{x,v'}{\longrightarrow}^{p} \subseteq \underset{x,v'}{\longrightarrow}^{p} \circ \underset{u,v}{\longrightarrow}^{2^{>p}} \quad \text{with } u, v' \notin X \tag{3.3}$$

$$\underset{x,v}{\longrightarrow}^{>p} \circ \underset{u',y}{\longrightarrow}^{p} \subseteq \underset{u',y}{\longrightarrow}^{p} \circ \underset{x,v}{\longrightarrow}^{2^{>p}} \cup \underset{x,v}{\longrightarrow}^{p} \circ \underset{u',y}{\longrightarrow}^{2^{>p}} \tag{3.4}$$

where $\longrightarrow^{>p}$ (resp. $\longrightarrow^{\geqslant p}$) denotes rewriting at any position greater than (resp. greater or equal to) p, and $\longrightarrow^{2^{>p}}$ or $\longrightarrow^{2^{\geqslant p}}$ denotes multi-step rewriting at any set of such positions. Let us now prove the first inclusion. Let

$$r \underset{u,v}{\longrightarrow}^{p} s \underset{x,v'}{\longrightarrow}^{q} t$$

with $p \geqslant q$. One can find 1-contexts $\bar{r}$ and $\bar{s}$ and substitutions ρ and σ such that $r = \bar{r}[u\rho]$, $s = \bar{r}[v\rho] = \bar{s}[x\sigma]$ and $t = \bar{s}[v'\sigma]$, with $pos(\Box, \bar{r}) = p$ and $pos(\Box, \bar{s}) = q$. As $p \geqslant q$ and by hypothesis on R, there is a substitution γ such that $\bar{r} = \bar{s}[x\gamma]$ and $\sigma = \gamma[v\rho]$. It is thus possible to swap the rewriting steps between r and t:

$$r = (\bar{s}[x\gamma])[u\rho] \underset{x,v'}{\longrightarrow}^{q} (\bar{s}[v'\gamma])[u\rho] \underset{u,v}{\longrightarrow}^{P} (\bar{s}[v'\gamma])[v\rho] = t,$$

where $P \subseteq 2^{>q}$ is the set of positions at which the special variable $\Box$ occurs in $\bar{s}[v'\gamma]$, i.e. $P = pos(\Box, \bar{s}[v'\gamma])$. Hence

$$r \underset{x,v'}{\longrightarrow}^{q} \circ \underset{u,v}{\longrightarrow}^{2^{>q}} t,$$

and inclusion (3.1) is proved. More generally, consider any two-steps rewriting

$$r \underset{u,v}{\longrightarrow}^{p} s \underset{u',v'}{\longrightarrow}^{q} t$$

with $p > q$. By definition there must exist 1-contexts $\bar{r}$ and $\bar{s}$ and substitutions ρ and σ such that $r = \bar{r}[u\rho],\ s = \bar{r}[v\rho] = \bar{s}[u'\sigma]$ and $t = \bar{s}[v'\sigma]$, with $pos(\Box, \bar{r}) = \{p\}$ and $pos(\Box, \bar{s}) = \{q\}$. The hypotheses we made on R imply certain restrictions on the structure of configuration s. As R is top-down, $\bar{s}[u'] \leqslant \bar{r}$. So there must be a substitution γ such that $\bar{r} = \bar{s}[u'\gamma]$ and $\sigma = \gamma[v\rho]$. Hence

$$r = (\bar{s}[u'\gamma])[u\rho] \underset{u',v'}{\longrightarrow}^{q} (\bar{s}[v'\gamma])[u\rho] \underset{u,v}{\longrightarrow}^{P} (\bar{s}[v'\gamma])[v\rho] = t,$$

where $P = pos(\Box, \bar{s}[v'\gamma])$. Thus

$$r \underset{u',v'}{\longrightarrow}^{q} \circ \underset{u,v}{\longrightarrow}^{P} t.$$

As $P \subseteq 2^{\geqslant q}$, inclusion (3.2) is proved. If $v' \notin X$ or $v' = y \in X$ and $\gamma(y)$ is not trivial, then $P \subseteq 2^{>q}$, so (3.3) and the first part of (3.4) are true. Finally, if $u \in X,\ v' \in X$ and $\gamma(v')$ is trivial, then $r = \bar{r}\rho,\ t = \bar{s}\sigma,\ \bar{r} = \bar{s}[u'],\ v\rho = \sigma$, hence

$$r = (\bar{s}[u'])\rho \underset{x,v}{\longrightarrow}^{q} (\bar{s}[v[u']])\rho \underset{u',x'}{\longrightarrow}^{P \subseteq 2^{>q}} (\bar{s}[v])\rho = t.$$

□

3.2.1.2 Rationality of Derivations

We are now ready to prove the rationality of the derivation of any top-down rewriting system. Using this property of top-down systems, it is possible to build a grammar which directly generates the derivation of any such system. This grammar mimics the way a rational word transducer works, using its control state to keep in memory a finite sub-term already read or yet to produce.

Theorem 3.14. *Every finite linear top-down term rewriting system R has a rational derivation.*

Proof. Let R be a finite linear top-down system. We denote by O the set of all overlappings between left and right parts of rules of R:

$$O = \{\ t \in C_n(F,X) \mid \exists\ s \in C_1(F,X), \\ u \in T(F,X)^n, s[t] \in Ran(R)\ \wedge\ t[u] \in Dom(R)\ \}. \quad (3.5)$$

Remark that $\Box$ belongs to O. We will now build a grammar G whose language is exactly the derivation of R. Its finite set of non-terminals is $\{<*>\} \cup Q$, where $Q = \{\ <t> = <t>_1 \ldots <t>_{n+1}\ \mid t \in O \cap C_n(F)\}$ and, for all $<t> \in Q$, $\pi_2(<t>)$ is a single variable. The production rules of G are of four types.
Type (1): $\forall\ f \in F_n$,

$$<\Box>\ \rightarrow\ f\ <\Box>_1^1 \ldots <\Box>_1^n\ \times\ f\ <\Box>_2^1 \ldots <\Box>_2^n$$
$$<*>\ \rightarrow\ f\ <*>^1 \ldots <*>^n$$

Type (2): $\forall\ t \in O \cap C_n(F),\ t[u] \in O \cap C_m(F)$,

$$<t>\ \rightarrow\ u[\pi_1(<t[u]>)]\ \times\ \pi_2(<t[u]>)$$

Type (3): $\forall\ t[u] \in O,\ t \in O \cap C_\ell(F)$ (necessarily $\{u_1, \ldots u_\ell\} \subseteq O$),

$$<t[u]>\ \rightarrow\ \pi_1(<u_1>) \ldots \pi_1(<u_\ell>) \times\ t[\pi_2(<u_1>) \ldots \pi_2(<u_\ell>)]$$

Type (4): $\forall\ (t[u], s[v]) \in R,\ v = v_1 \ldots v_\ell$ (necessarily $\{t, v_1, \ldots v_\ell\} \subseteq O$),

$$<t>\ \rightarrow\ u\sigma\ \times\ s[\pi_2(<v_1>) \ldots \pi_2(<v_\ell>)]$$

where σ is a variable renaming such that for any variable x of u, $\sigma(x) = <v_i>_j$ if x is the j-th variable to appear in v_i (from left to right), and $\sigma(x) = <*>$ if x does not appear in any of the v_i. Figure 3.2 illustrates the four types of rules.

Intuitively, the role of this substitution is to gather into the same non-terminal or hyper-edge all the variables of u belonging to the same v_i, while respecting the order in which these variables appear in v_i. This way, a correct instantiation of non-terminals of G is ensured. If a variable of u does not appear at all in v, then it means that a whole input subtree is 'discarded' by the rewriting rule being applied. Thus the grammar should accept any subtree to be generated at this position, which is the role of the unary non-terminal $<*>$.

For simplicity, we only consider type (4) rules in which $t, v_1, \ldots v_\ell$ are *maximal*. The other cases can be simulated by suitable finite compositions of rules of types (2), (3) and (4). $\square$

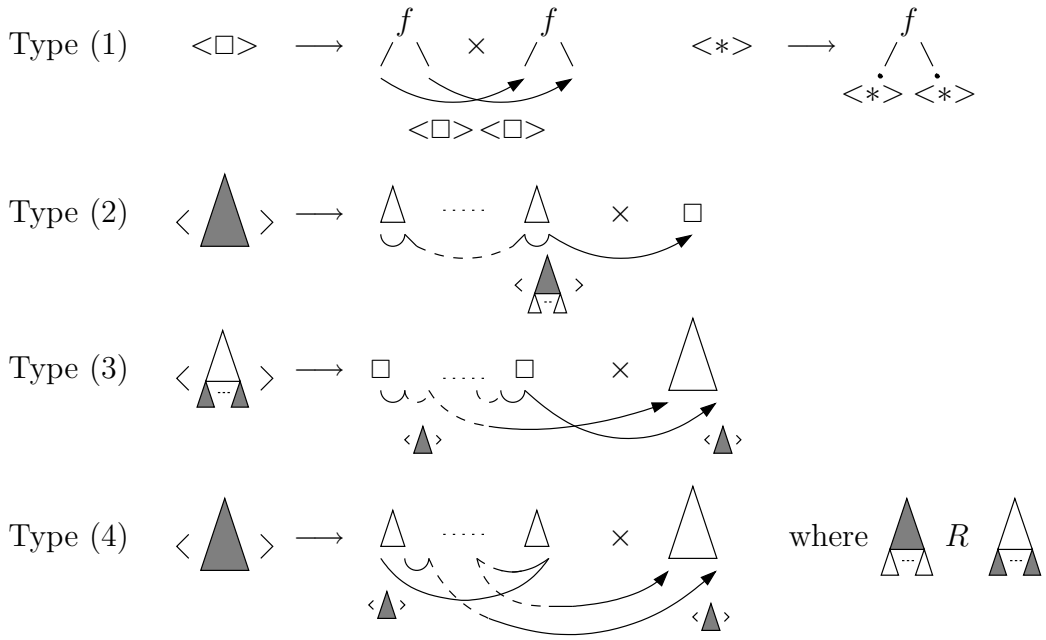


Figure 3.2: Grammar associated to a top-down system.

Remark 3.15. It should be possible to extend this proof without too much difficulty to top-down systems with a recognizable set of rules: instead of being finite, the set O of overlapping parts between left and right-hand sides of rules would be a regular set. It would then be sufficient, instead of associating an element in O to each non-terminal in the grammar, to associate non-terminals to control states of a top-down tree automaton accepting O.

Example 3.16. Consider the linear top-down system R over the alphabet $F = \{f^{(2)}, g^{(1)}, h^{(1)}, a^{(0)}\}$ with a unique rule $fgxgy \to hfxy$. The corresponding grammar is the grammar of Ex. 3.11 where each non-terminal stands for one of the possible overlappings of rules of R: A stands for $\Box$ and B for $f\Box_1\Box_2$. Note that type (4) rules with non-maximal overlappings have been discarded. This example also illustrates the fact that the inverse image of a regular term language by the derivation of a linear top-down system is not regular in general: for instance, the image by R_G^{-1} of h^*faa is $\{h^*fg^nag^na \mid n \geq 0\}$, which is not regular.

3.2.1.3 Regularity Preservation

We will now mention a property of top-down systems, which has been known for the past few years for larger classes of systems. We nevertheless propose a direct construction for our case.

Proposition 3.17. *The image of any regular term language by the derivation relation of a finite linear top-down term rewriting system is regular.*

Proof. First note that the domain of the derivation of a top-down system is $T(F)$, and its range is regular: the grammar constructed in the proof of Theorem 3.14 only has non-terminals whose second projection is reduced to a single variable.

Let L be a regular term language accepted by some top-down tree automaton A with a set of control states Q. Let R be a linear descending system. By the previous construction, we are able to build a rational grammar G recognizing $\xrightarrow[R]{*}$. Let us now define a 'product grammar' G_A whose domain is L and whose range is $\xrightarrow[R]{*}(L)$. G_A's non-terminals will be of the form

$$(N_1, q_1) \dots (N_n, q_n)(N_{n+1}, q_1 \dots q_n), \tag{3.6}$$

noted $N_{q_1 \dots q_n}$ for short. For each production

$$N_1 \dots N_{n+1} \longrightarrow t_1 \dots t_n \; s$$

of G, the product grammar G_A will have all possible productions

$$N_{q_1 \dots q_n} \longrightarrow t'_1 \dots t'_n \; s'$$

where the m-contextual term word $t_1 \dots t_n$ is partially accepted by A (see Section 1.3.5.2) with initial control word $(q_1 \dots q_n)$ and final control word $(q'_1 \dots q'_m)$. Furthermore, $t'_1 \dots t'_n$ is obtained from $t_1 \dots t_n$ by pairing each of its m variables with the associated component of the final control word, and s' is obtained from s by pairing each of its variables with a word on Q^*, so as to complete every instance of G_A's non-terminals according to (3.6). The result of this is that a pair (s, t) belongs to $L(G_A)$ if and only if $(s, t) \in L(G)$ and $s \in L$. In other words, the second projection of G_A is exactly the set of terms who are the image of some term in L by $\xrightarrow[R]{*}$, so $\pi_2(L(G_A)) = \xrightarrow[R]{*}(L)$. By forgetting the left projection of every grammar production, one gets a grammar where all non-terminals have an arity of 1. Such grammars are called *regular tree grammars* and generate regular term languages. In this case, we obtain a regular grammar generating $\xrightarrow[R]{*}(L)$. □

The inverse of a top-down system is, by definition, bottom-up. For any top-down system we can build a grammar G recognizing $\xrightarrow[R]{*}$. Thus, the grammar $\pi_2(G)\pi_1(G)$ obtained by swapping both projections of G generates the derivation $\xrightarrow[R^{-1}]{*}$ of the bottom-up system R^{-1}. Inverse recognizability preservation follows.

Corollary 3.18. *Every finite linear bottom-up term rewriting system R has a rational derivation $\xrightarrow[R]{*}$, and the inverse image by $\xrightarrow[R]{*}$ of any regular term language is regular.*

The question of finding classes of term-rewriting systems which preserve recognizability of term languages has been addressed by several authors. Top-down systems form a strict subfamily of *generalized semi-monadic term rewriting systems* [GV98], which is itself a strict subfamily of *right-linear finite path overlapping systems* [TKS00]. Both classes have been proved to preserve recognizability. As a consequence, this is also the case for top-down systems. However, it should be mentioned that neither of these classes has a rational derivation. For instance, the generalized semi-monadic system whose unique rule is $gx \longrightarrow fgfx$ clearly has a non-rational derivation: its intersection with the rational relation $ga \times f^*gf^*a$ is $ga \times \{f^ngf^na | n \geq 0\}$. By the usual pumping arguments (adapted to this new setting), this relation is not rational.

3.2.2 Prefix Systems

Root rewriting systems are already known to be very powerful: indeed, they can simulate the execution steps of Turing machines. This implies a direct negative result concerning their derivations.

Proposition 3.19. *At least one linear prefix tree rewriting system has a non recursive derivation.*

Proof. Let $M = (\Sigma, \Gamma, [,], Q, q_0, F, \delta)$ be a Turing machine with only ε-rules, let us build a prefix system R_M over the alphabet $Q \cup \Gamma \cup \{[,], \bot\}$, with variables in $\{x, y\}$, where symbols in Q are considered binary, symbols in $\Gamma \cup \{[,]\}$ are unary and $\bot \notin \Gamma$ is a constant. We translate the rules of M into rewriting rules of R_M as follows:

$$
\begin{array}{ll}
\forall pA \xrightarrow{\varepsilon} qB+ \in \delta, & (pxAy, qBxy) \in R_M \\
\forall pA \xrightarrow{\varepsilon} qB \in \delta, & (pxAy, qxBy) \in R_M \\
\forall pA \xrightarrow{\varepsilon} qB- \in \delta,\ C \in (\Gamma \cup \{[\,\}), & (pCxAy, qBCxy) \in R_M \\
\forall pA \xrightarrow{\varepsilon} qBA \in \delta, & (pxAy, qxBAy) \in R_M \\
\forall pA \xrightarrow{\varepsilon} q \in \delta, & (pxAy, qxy) \in R_M
\end{array}
$$

This system has both overlappings of the kind $l = r\sigma$ and of the kind $r = l\sigma$, for some left and right-hand sides l and r and substitution σ. It is thus a prefix system, and is neither top-down, bottom-up or suffix in general. It is quite clear that computing the derivation of R_M is at least as difficult as computing the

reachability relation of M, and is thus undecidable. Hence $\xrightarrow[R]{*}$ is non-recursive and can obviously not be rational. □

3.2.3 Suffix Systems

The family of ground rewriting systems, which is a sub-family of suffix systems, has already been studied by several authors. In particular, Dauchet and Tison [DT85] showed that the derivations of ground systems can be recognized by a certain type of composite automata called *ground tree transducers* (GTT). We use similar arguments in order to prove that, more generally, any suffix system has a rational derivation. After introducing a property related to the notion of *suffix rewriting*, we show that the derivation of any recognizable linear suffix system is rational. Finally, we prove that the image or inverse image of any regular term language by the derivation of a recognizable linear suffix system is regular, and that it is possible to build a tree automaton accepting it.

3.2.3.1 Suffix Rewriting

Ground rewriting systems are systems whose rules do not contain any variables. As a consequence, they can only rewrite ground sub-terms. Ground rewriting rules can be seen as a kind of pushdown rules on a tree-like stack, where the only parts one is entitled to modify are the areas around the leaves.

Given a non-ground term rewriting system R over F with variables in X, we define a restricted notion of rewriting by R over terms in $T(F \cup Y)$, where Y is an arbitrary set of variables. This notion, inspired by ground rewriting and called suffix rewriting, considers variables in Y temporarily as constants, while substitution of variables occurring in the rules of R is restricted to elements of Y. We call this suffix rewriting, since such a restriction only allows rewriting rules to be used "near" the constants and variables.

Definition 3.20. For all rewriting system R over F with variables in X, and all set of variables Y disjoint from F_0, we define the *suffix rewriting* relation of R as

$$\xrightarrow[R]{} \;=\; \left\{ (c[l\sigma], c[r\sigma]) \in T(F \cup X)^2 \mid (l, r) \in R \wedge \; c \in C_1(F) \wedge \; \sigma \in Y^X \right\}$$

where Y^X is the set of mappings (or ground substitutions) from X to Y.

The reason why we do not simply treat Y as a set of constants, which would indeed define a ground system over $F \cup Y$, is that we wish to consider two terms in $T(F \cup Y)$ as equal if they differ only by a renaming of their "constant variables" (constants in Y).

3.2.3.2 Decomposing Derivations

Suffix systems have a specific behavior with respect to suffix rewriting. Indeed, the derivation of any input tree t by a suffix system can always be decomposed in two phases. First, a prefix $\bar{t}$ of t is read, and several steps of suffix rewriting can be applied to it. Once this first sequence is over, $\bar{t}$ has been rewritten into a prefix $\bar{s}$ of s, never to be modified anymore. In a second time, the rest of t is derived in the same fashion, starting with suffix rewriting of a prefix of the remaining input. As a consequence, the derivation of a suffix system is equivalent to its 'iterated' suffix derivation.

Lemma 3.21. *For any suffix term rewriting system R,*

$$s \xrightarrow[R]{k>0} t \iff \begin{cases} \exists\ \bar{s}, \bar{t} \in T(F,X),\ \sigma, \tau \in S(F,X),\ k' > 0,\ s = \bar{s}\sigma\ \wedge\ t = \bar{t}\tau \\ \quad \wedge\ \bar{s} \xrightarrow[R]{k'} \bar{t}\ \wedge\ \forall x \in Var(\bar{s}) \cap Var(\bar{t}),\ \sigma(x) \xrightarrow[R]{\leq k-k'} \tau(x). \end{cases}$$

Proof. First note that, since $s \xrightarrow[R]{} t \implies s\sigma \xrightarrow[R]{} t\sigma$ for all ground substitution σ, the inverse implication is trivial. Let us prove the direct implication by induction on k:

$k = 1$: $s \xrightarrow[R]{} t$ implies that there is a context c and a substitution σ such that $s = c[l\sigma]$ and $t = c[r\sigma]$. Thus by definition of suffix rewriting $c[l] \xrightarrow[R]{} c[r]$, and of course for all variable x, $\sigma(x) \xrightarrow[R]{0} \sigma(x)$.

$k \Rightarrow k+1$: let $s \xrightarrow[R]{k} s' \xrightarrow[l,r]{p} t$ with lRr, $s = \bar{s}\sigma$ and $s' = \bar{s'}\sigma'$. Two cases:

- $p \in Pos(\bar{s'})$: if there exists a context c such that $\bar{s'} = c[l]$, then we have $\bar{s} \xrightarrow[R]{k} \bar{s'} \xrightarrow[R]{} c[r]$ and the condition is verified with $k' = k$ and $t = (c[r])\sigma'$. If not, then there must exist a context c and a non-trivial substitution ω' such that $\bar{s'} = (c[l])\omega'$. As R is suffix and as, by induction hypothesis, $\bar{s} \xrightarrow[R]{*} \bar{s'}$, there must exist ω such that $\bar{s} = (c[l])\omega$ and for all variable x common to l and r, $\omega(x) \xrightarrow[R]{*} \omega'(x)$. We can then write s as $(c[l])\omega\sigma$, t as $(c[r])\omega'\sigma'$, and verify the condition is true with $k' = 1$.
- $p \notin Pos(\bar{s'})$: by induction hypothesis, one can find $k' > 0$ such that for all x common to $\bar{s}$ and $\bar{s'}$, $\sigma(x) \xrightarrow[R]{\leq k-k'} \sigma'(x)$. Furthermore, by applying rule (l, r) to one of the $\sigma'(x)$, we get $\sigma'(x) \xrightarrow[l,r]{} \tau(x)$. We thus have $t = \bar{s'}\tau$,

$\bar{s} \xrightarrow[R]{k'} \bar{s'}$ and $\sigma(x) \xrightarrow[R]{k-k'+1} \tau(x)$ for all x in both $\bar{s}$ and $\bar{s'}$, which verifies the condition and concludes the proof.

□

Another interesting property is that, for any linear recognizable system, a suffix rewriting sequence is always equivalent to a sequence in two parts, where the first part only consumes suffix sub-terms of the input term, and the second part only produces new suffix sub-terms in their place.

Lemma 3.22. *For all recognizable linear term rewriting system R over F and X, there exist a finite ranked alphabet Q and three finite rewriting systems*

- $R_- \subseteq \{px \longrightarrow fp_1x_1 \dots p_nx_n \mid f \in F,\ p, p_1, \dots, p_n \in Q,\ x, x_1, \dots, x_n \in X^*\}$
- $R_= \subseteq \{px \longrightarrow qy \mid p, q \in Q,\ x, y \in X^*\}$
- $R_+ \subseteq \{fp_1x_1 \dots p_nx_n \longrightarrow px \mid f \in F,\ p, p_1, \dots, p_n \in Q,\ x, x_1, \dots, x_n \in X^*\}$

such that $s \xrightarrow[R]{*} t \iff s \xrightarrow[R_+ \cup R_=]{*} \circ \xrightarrow[R_- \cup R_=]{*} t$.

Proof. Let R be a recognizable linear rewriting system over F and X. Let $(A_i, B_i)_{i \in [1,l]}$ be l pairs of finite top-down automata, each accepting a set of rules of R. The set of states of A_i (resp. B_i) will be referred to as P_i (resp. Q_i). Without losing generality, we will suppose that all P_i and Q_i are pairwise disjoint. We also define P as the union of all P_i and Q as the union of all Q_i. For all state $p \in P \cup Q$, define $\nu(p)$ as the set of all possible variable boundaries (i.e. tuples of variables appearing at the leaves of a term, read from left to right) in the language accepted with initial state p. The way we defined recognizable linear systems, i.e. with a finite number of variables, we can assume that $|\nu(p)| = 1$ for all p.

In a first step, we define a new rewriting system R' on $F \cup P \cup Q$ and X. The state alphabets P and Q are considered as ranked alphabets, where the arity of any of their symbols is equal to the number of variables appearing in the terms of its associated term language (which can be supposed unique without losing generality). For instance, suppose that from some state p, automaton A accepts the language g^*fxy. Then, p will be considered as a binary symbol.

We give R' the following set of rules. For all rule $pf \to p_1 \dots p_m$ in some A_i such that $\nu(p) = \nu(p_1) \dots \nu(p_m) = x_1 \dots x_n$, we have:

$$fp_1\nu(p_1) \dots p_m\nu(p_m) \ \to \ px_1 \dots x_n \ \in \ R'.$$

Rules of this kind allow us to consume a left-hand side of a rule in the input tree. For all rule $qf \to q_1 \dots q_m$ in some B_i such that $\nu(q) = \nu(q_1) \dots \nu(q_m) = x_1 \dots x_n$, we have:

$$qx_1 \dots x_n \ \to \ fq_1\nu(q_1) \dots q_m\nu(q_m) \ \in \ R'.$$

Rules of this kind allow us to produce a right-hand side of a rule whose left-hand side has been previously consumed. Notice that, in these last two cases, if $f = x$ for some variable x we obtain rules of the form $x \to px$ and $qx \to x$ respectively. Finally, for all pair (p_0, q_0) of initial states of some pair (A_i, B_i), with $\nu(p_0) = x_1 \dots x_n$ and $\nu(q_0) = x_{k_1} \dots x_{k_m}$ we have:

$$p_0x_1 \dots x_n \ \to \ q_0x_{k_1} \dots x_{k_m} \ \in \ R'.$$

This simulates the application of a rewriting rule from $L(A_i) \times L(B_i)$ by initiating a run of automaton B_i when a successful 'reverse run' of A_i has been achieved. When restricted to $T(F)^2$, the derivation of R' coincides with $\xrightarrow[R]{*}$:

$$\forall s, t \in T(F), \ s \xrightarrow[R]{*} t \iff s \xrightarrow[R']{*} t. \tag{3.7}$$

The proof of this property is not difficult and will thus not be detailed here. In the rest of the proof, p, q and all variations thereof designate automata control states in $P \cup Q$, tuples of variable in X^* are denoted by $u, v, u_i, v_i, \dots$, and σ is a variable renaming.

In a second step, we define R_+ as $\{fp_1u_1 \dots p_nu_n \, R' \, pu\}$ ('consuming' rules), R_- as $\{qv \, R' \, fq_1v_1 \dots q_nv_n\}$ ('producing' rules) and $R_=$ as the smallest binary relation in $T(F, X)^2$ closed under the following inference rules:

$$\frac{}{pu \ R_= \ pu} \quad (1) \qquad \frac{pu \ R' \ qv}{pu \ R_= \ qv} \quad (2)$$

$$\frac{pu \ R_= \ qv}{pu\sigma \ R_= \ qv\sigma} \quad (3) \qquad \frac{qu \ R_= \ q'v \quad q'v \ R_= \ q''z}{qu \ R_= \ q''z} \quad (4)$$

$$\frac{pu \ R_- \ fp_1u_1 \dots p_nu_n \quad fq_1v_1 \dots q_nv_n \ R_+ \ qv \quad \forall i, \, p_iu_i \ R_= \ q_iv_i}{pu \ R_= \ qv} \quad (5)$$

Since F, P, Q and X are finite, and each symbol p of $P \cup Q$ has a well-defined arity, then $R_=$ is finite and effectively computable. Let us mention a simple property of $R_=$:

$$\forall \, pu, t, qv \in T(F, X), \ pu \xrightarrow[R_- \cup R_=]{*} t \xrightarrow[R_+ \cup R_=]{*} qv \implies pu \, R_= \, qv. \tag{3.8}$$

This can be proved by induction on the height k of term t:

$k = 0$: no rule of R_- or R_+ is applied, thus $pu \xrightarrow[R_=]{*} qv$. Then, by inference rule (4), the property is true.

$k \Rightarrow k+1$: let us decompose the derivation sequence between s and t:

$$pu \xrightarrow[R_=]{} p'u' \xrightarrow[R_-]{} fp_1u_1 \dots p_nu_n$$
$$\xrightarrow[R_-\cup R_=]{*} \circ \xrightarrow[R_+\cup R_=]{*} fq_1v_1 \dots q_nv_n \xrightarrow[R_+]{} q'v' \xrightarrow[R_=]{} qv.$$

By induction hypothesis, $fp_1u_1 \dots p_nu_n \xrightarrow[R_=]{*} fq_1v_1 \dots q_nv_n$, so by inference rule (5), $p'u' \, R_= \, q'v'$. Finally, by inference rule (4), $pu \, R_= \, qv$.

It remains to prove that R_+, $R_=$ and R_- verify the lemma. By (3.7), it suffices to show that $\xrightarrow[R_+\cup R_=]{*} \circ \xrightarrow[R_-\cup R_=]{*} = \xrightarrow[R']{*}$. Proving the direct inclusion is equivalent to proving $\xrightarrow[R_=]{} \subseteq \xrightarrow[R']{*}$. Suppose $s \xrightarrow{R_=} t$ for some terms s and t. The rule of $R_=$ used in this rewriting step can only be defined using the inference rules above. We can thus reason by induction on the sequence of inference rules used to define it to show that it must entail $s \xrightarrow[R']{*} t$. This is pretty straightforward and will not be detailed here.

To prove the converse we will establish, by induction on k, the following property:

$$\forall k,\ s \xrightarrow[R']{k} t \implies \exists\, c \in C_1(F),\ (q_iu_i)_{i\in[n]} \in T(F,X),$$
$$s \xrightarrow[R_+\cup R_=]{*} c[q_1u_1 \dots q_nu_n] \xrightarrow[R_-\cup R_=]{*} t.$$

$k = 0$: $s = t = c$, $n = 0$, so trivially $s \xrightarrow[R_+\cup R_=]{*} c \xrightarrow[R_-\cup R_=]{*} t$.

$k \Rightarrow k+1$: let $s \xrightarrow[R']{k} t \xrightarrow[l,r]{} t'$ with $lR'r$. By induction hypothesis,

$$s \xrightarrow[R_+\cup R_=]{*} c[q_1u_1 \dots q_nu_n] \xrightarrow[R_-\cup R_=]{*} t = c[t_1 \dots t_n].$$

There are three possible cases:

- $\exists\, c',\ t = c'[t_1 \dots t_n\, l]$: in this case s can be written $c'[s_1 \dots s_n\, l]$, so the

following derivation sequence is valid:

$$\begin{aligned} s &\xrightarrow[R_+\cup R_=]{*} c'[q_1u_1 \dots q_nu_n\, l] \xrightarrow[R_+]{} c'[q_1u_1 \dots q_nu_n\, q_lu] \\ &\xrightarrow[R_=]{} c'[q_1u_1 \dots q_nu_n\, q_rv] \text{ (because } l\, R'\, r) \\ &\xrightarrow[R_-]{} c'[q_1u_1 \dots q_nu_n\, r] \xrightarrow[R_-\cup R_=]{*} c'[t_1 \dots t_n\, r] = t'. \end{aligned}$$

- $\exists\, i,\ t_i = \bar{t}_i[l]$: the only way to produce t_i from q_iu_i is along the steps

$$q_iu_i \xrightarrow[R_-\cup R_=]{*} \bar{t}_i[q_lu] \xrightarrow[R_-]{} t_i.$$

Thus the following derivation is valid:

$$\begin{aligned} s &\xrightarrow[R_+\cup R_=]{*} c[q_1u_1 \dots q_nu_n] \xrightarrow[R_-\cup R_=]{*} c[q_1u_1 \dots \bar{t}_i[q_lu] \dots q_nu_n] \\ &\xrightarrow[R_=]{} c[q_1u_1 \dots \bar{t}_i[q_rv] \dots q_nu_n] \xrightarrow[R_-]{} t'. \end{aligned}$$

- $\exists\, c',\ j > i \geq 1,\ t = c'[t_1 \dots t_i\, l\, t_{j+1} \dots t_n]$, with $l = \bar{l}[t_{i+1} \dots t_j]$. The derivation sequence between s and t' can then be written

$$\begin{aligned} s &\xrightarrow[R_+\cup R_=]{*} c[q_1u_1 \dots q_nu_n] \xrightarrow[R_-\cup R_=]{*} c[t_1 \dots t_n] \\ &\xrightarrow[R_+]{*} c[t_1 \dots t_i\, q'_{i+i}u'_{i+1} \dots q'_ju'_j\, t_{j+1} \dots t_n] \\ &\xrightarrow[R_+]{} c'[t_1 \dots t_i\, q_lu\, t_{j+1} \dots t_n] \xrightarrow[R_=]{} c'[t_1 \dots t_i\, q_rv\, t_{j+1} \dots t_n] \\ &\xrightarrow[R_-]{} c'[t_1 \dots t_i\, r\, t_{j+1} \dots t_n] = t' \end{aligned}$$

Notice that for all $k \in [i+1, j]$, $q_ku_k \xrightarrow[R_-\cup R_=]{*} t_k \xrightarrow[R_+\cup R_=]{*} q'_ku'_k$. Thus, by (3.8), $q_ku_k\, R_=\, q'_ku'_k$, hence

$$\begin{aligned} s &\xrightarrow[R_+\cup R_=]{*} c[q_1u_1 \dots q_nu_n] \xrightarrow[R_+\cup R_=]{*} c'[q_1u_1 \dots t_i\, q_lu\, q_{j+1}u_{j+1} \dots q_nu_n] \\ &\xrightarrow[R_-\cup R_=]{*} c'[t_1 \dots t_i\, r\, t_{j+1} \dots t_n] = t'. \end{aligned}$$

This concludes the proof of lemma 3.22. □

Lemma 3.22 can be reformulated in the following way: a pair (s, t) of terms belongs to the suffix derivation of a system R if and only if there is a context c such that $s = c[s_1 \dots s_n]$, $t = c[t_1 \dots t_n]$ and for all $i \in [1, n]$, there is a term q_ix_i such that $s_i \xrightarrow[R_+\cup R_=]{*} q_ix_i$ and $q_ix_i \xrightarrow[R_-\cup R_=]{*} t_i$.

3.2.3.3 Rationality of Derivations

Now that the structure of derivations of suffix systems is better understood, we are able to build a grammar generating the derivation relation of any suffix system.

Theorem 3.23. *Every recognizable linear suffix term rewriting system R has a rational derivation.*

Proof. Let R be a recognizable linear suffix system on $T(F, X)$. Let R_+, $R_=$ and R_- be the rewriting systems mentioned in Lemma 3.22. Let N be a set of pairs of the form $u|v$ where u and v are two linear term words over $Ran(R_+ \cup R_=)^*$ and $Dom(R_- \cup R_=)^*$ respectively. Note that Ran and Dom are defined up to a renaming of the variables. We can thus impose that u and v share the same set of variables ($Var(u) = Var(v)$), and there is no pair of strict sub-words u' and v' of u and v such that $Var(u') \neq Var(v')$ (i.e one should not be able to split $u|v$ in two correct non-terminals). This, together with the facts that F is finite and u and v are linear, implies that N is finite for some fixed, standard variable renaming. Thus, given an axiom I, we can build a grammar G whose set of non-terminals is $N \cup \{I, I'\}$, having the following finite sets of productions:

$$\forall\ f \in F,\qquad I \longrightarrow f I_1^1 \dots I_1^n \times f I_2^1 \dots I_2^n \quad \text{and} \quad I' \longrightarrow f I'^1 \dots I'^n \tag{3.9}$$

$$\forall\ px \in Dom(R_- \cup R_=) \cap Ran(R_+ \cup R_=),\qquad I \longrightarrow px|px \tag{3.10}$$

$$\forall\ u' \xrightarrow[R_=]{*} u,\quad v \xrightarrow[R_=]{*} v',\quad u' \in Ran(R_+)^*,\quad v' \in Dom(R_-)^*,\qquad u|v \longrightarrow u'|v' \tag{3.11}$$

$$\forall\ u_1 = p_1x_1 \dots p_ix_i,\quad u_2 = p_{j+1}x_{j+1} \dots p_nx_n,\quad v = q_1y_1 \dots q_my_m,\quad f p_{i+1}x_{i+1} \dots p_jx_j\, R_+\, px,\quad u_1\, px\, u_2|v \longrightarrow \mu_1 \dots \mu_i\, (f \mu_{i+1} \dots \mu_j)\, \mu_{j+1} \dots \mu_n \times \nu_1 \dots \nu_m \tag{3.12}$$

$$\forall\ u = p_1x_1 \dots p_nx_n,\quad v_1 = q_1y_1 \dots q_iy_i,\quad v_2 = q_{j+1}y_{j+1} \dots q_my_m,\quad qy\, R_-\, f q_{i+1}y_{i+1} \dots q_jy_j,\quad u|v_1\, qy\, v_2 \longrightarrow \mu_1 \dots \mu_n \times \nu_1 \dots \nu_i\, (f \nu_{i+1} \dots \nu_j)\, \nu_{j+1} \dots \nu_m \tag{3.13}$$

In rules (3.12) and (3.13), all the $(\mu_k)_{k\in[1,n]}$ and $(\nu_k)_{k\in[1,m]}$ are variables belonging to instances of non-terminals $u'|v' \in N$ where u' and v' are built from terms $(p_kx_k)_{k\in[1,n]}$ and $(q_ky_k)_{k\in[1,m]}$ respectively. Variables μ_1 to μ_n (resp. ν_1 to ν_m) appear only in the first (resp. second) projection of any non-terminal. Note that

this instantiation is unique, by construction of the set N. It is also always possible since every rule of R is, by hypothesis, linear.

Call ρ the substitution which maps each non-terminal variable $(u|v)_i$ to the term $(u)_i$ if $i \in [1, |u|]$ and to $(v)_i$ if $i \in [|u|+1, |u|+|v|]$, and each non-terminal variable $(I^i_j)_{j\in[1,2]}$ to a variable x_i. It is clear from the rules of G_0 that:

$$I \xrightarrow[G_0]{*} s \times t \iff s\rho \xrightarrow[R_+\cup R_=]{*} \circ \xrightarrow[R_-\cup R_=]{*} t\rho. \tag{3.14}$$

We will not detail the proof of this observation. Notice that this grammar works in a very similar way to a *ground tree transducer*, which is the formalism used by [DT85] to generate the derivation of a ground system. The only difference is that we keep track of the variables appearing in the left and right projections of the relation, so as to be able to resume the rewriting at relevant positions. Now add to G_0 the set of rules

$$\forall\, x \in X \text{ such that } xRpx,\ qxRx, \qquad px|qx \longrightarrow I \tag{3.15}$$

$$px| \longrightarrow I' \tag{3.16}$$

and call this new grammar G. These last rules allow the derivation to go on properly after a first sequence of suffix rewritings has taken place, by creating new instances of the axiom between leaves where the same variable would appear. By Lemma 3.21, G generates $\xrightarrow[R]{*}$. □

Remark 3.24. Note that the grammar constructed in the previous proof matches the definition of rational transductions (cf. Def. 3.12. The derivations of suffix systems are thus a subfamily of rational transductions.

3.2.3.4 Regularity Preservation

Unlike top-down systems, suffix systems preserve the regularity of term languages both by forward and inverse application.

Proposition 3.25. *The image and inverse image of any regular term language by the derivation of a recognizable linear suffix term rewriting system are regular.*

Proof. Let R be a recognizable linear suffix term rewriting system on $T(F)$, G the grammar recognizing its derivation, built as in the proof to Theorem 3.23, and $N \cup \{I, I'\}$ its set of non-terminals. Let A be a finite non-deterministic top-down tree automaton accepting a regular language L. Suppose Q_A is the set of control states of A, disjoint from F and X, $q_0 \in Q_A$ its unique initial state. Let Q'_A be a disjoint copy of Q_A, we define the following grammar G_A having non-terminals in $Q_A \cup Q'_A \cup \{(r_1, u_1)\dots(r_n, u_n)|v \mid r_1, \dots, r_n \in Q_A \wedge u_1 \dots u_n|v \in N\}$ and the following set of production rules:

- For all rule $rf \underset{A}{\longrightarrow} fr_1 \dots r_n$:

$$r \longrightarrow f(r_1)_1 \dots (r_n)_1 \times f(r_1)_2 \dots (r_n)_2 \tag{3.17}$$
$$r' \longrightarrow fr'_1 \dots r'_n \tag{3.18}$$

- For all $r \in Q_A$, $px|px \in N$:

$$r \longrightarrow (r, px)|px, \tag{3.19}$$

- For all rule $p_1x_1 \dots p_nx_n|v \longrightarrow p'_1x'_1 \dots p'_nx'_n|v'$ of type (3.11) in G and state word $r_1 \dots r_n \in Q_A^*$:

$$(r_1, p_1x_1) \dots (r_n, p_nx_n)|v \longrightarrow (r_1, p'_1x'_1) \dots (r_n, p'_nx'_n)|v' \tag{3.20}$$

- For all rule $u_1\, px\, u_2|v \longrightarrow s \times t$ of type (3.12) of G with $u_1 = p_1x_1 \dots p_ix_i$ and $u_2 = p_{j+1}x_{j+1} \dots p_nx_n$, and for all states word $r_1 \dots r_i\, r\, r_{j+1} \dots r_n \in Q_A^*$:

$$(r_1, p_1x_1) \dots (r_i, p_ix_i)(r, px)(r_j, p_jx_j) \dots (r_n, p_nx_n)|v \longrightarrow s' \times t' \tag{3.21}$$

  where $rf \underset{A}{\longrightarrow} fr_{i+1} \dots r_j$ and s' and t' are obtained from s and t by replacing each occurrence of p_k in a non-terminal variable by (r_k, p_k) for all $k \in [1, n]$.
- For all rule $p_1x_1 \dots p_nx_n|v \longrightarrow s \times t$ of type (3.13) of G and word $r_1 \dots r_n \in Q_A^n$:

$$(r_1, p_1) \dots (r_n, p_n)|v \longrightarrow s' \times t' \tag{3.22}$$

  where s' and t' are obtained from s and t by replacing each occurrence of p_k in a non-terminal variable by (r_k, p_k) for all $k \in [1, n]$.
- Finally, for all rule of the form $px|qx \longrightarrow I$ (resp. $px| \longrightarrow I'$) in G and all $r \in Q_A$:

$$(r, px)|qx \longrightarrow r \tag{3.23}$$
$$(r, px)| \longrightarrow r'. \tag{3.24}$$

One can see that, starting from non-terminal q_0, grammar G_A only accepts pairs (s, t) such that $s \underset{R}{\overset{*}{\longrightarrow}} t$ and $s \in L$. Thus the set of all t such that (s, t) is generated by G_A from q_0 is exactly the image of L by the derivation of R:

$$L(G_A, q_0) = \underset{R}{\overset{*}{\longrightarrow}} (L).$$

An automaton recognizing this set of terms can be built by taking the right projection of G_A, and by treating each non-terminal variable as a unary non-terminal. The rules of this automaton are given by the rules of G_A broken into several rules over these non-terminals of length 1.

Note that this proof is totally symmetrical, and that the synchronization of G by a finite automaton A could be done on the second projection instead of the first. Thus the converse result. □

3.3 Summary and perspectives

This work extends the left, right, prefix and suffix word rewriting systems defined in [Cau00] to bottom-up, top-down, suffix and prefix term rewriting systems. The derivation relation of the three first types of systems can be generated by finite graph grammars, while systems of the fourth type have a non recursive derivation in general. We also stated some recognizability preservation properties of these classes of systems, and provided effective constructions in each case. We also stated some regularity preservation properties of these classes of systems, and provided effective constructions in each case. This is however quite preliminary work, for which several further potential developments remain, some of which we will now briefly describe.

3.3.1 Classes of Relations over Terms

This study puts in practical use the notion of rationality defined in [Rao97], which nicely extends the usual rational relations on words, even though some of their key properties are missing, like the closure under composition or systematic preservation of recognizability. However, this formalism is an interesting and powerful work basis for the study of binary relations on terms, especially thanks to the fact that it is general enough to extend asynchronous transducers (which is not the case of most other formalisms). Still, depending on one's objectives, it might be necessary to devise a more restricted notion of rational relations on terms, which would be closed under composition or preserve recognizability (or both). Note that [Rao97] contains the definition of such a subfamily of relations (called rational *transductions*). However, it can be shown that the derivations of some top-down systems do not belong to this class.

A more in-depth study of the classes of derivations of top-down and suffix systems deserves to be led. In particular, it would be interesting to determine precise automata-theoretic characterizations of these classes of relations.

In the case of words, the derivations of left systems (the restriction of top-down systems to words) coincide with the class of rational relations. It seems possible to define a class of transducers characterizing exactly the derivations of top-down systems, and whose restriction to words yields precisely the class of rational relations. The main idea is to consider ranked sets of control states instead of unary control states, and to provide transition rules of the form

$$qs \to tq_1x_1 \dots q_nx_n$$

where q is a n-ary control state, s a term word of length n, t a term and $x_1 \dots x_n$ are tuples of variables occurring in s. Such transducers are syntactically more

general than the existing class of top-down transducers[2]. A condition for this study to be worthwhile would be to investigate other closure properties. A negative point is the already stated fact that the inverse image of a regular set of terms by such a relation may not be regular. One important remaining question is whether they are closed under composition, and whether this class can be restricted further while still generalizing rational word relations.

As for the derivations of suffix systems, their closure under composition is a consequence of [Rao97], since they belong to the class of rational tree transductions. Other properties, like their closure under boolean operations, should be investigated.

3.3.2 Top-Down and Suffix Rewriting Graphs

These results participate in the systematic study and understanding of families of finitely presented binary relations over infinite domains, in the spirit of previous lines of research on infinite graphs. In particular, they provide an immediate definition of several families of recursive graphs (in the sense that the existence of edges is decidable), which should be carefully compared to existing families.

Top-down derivation graphs extend rational graphs over words, and term-synchronized (or term-automatic) graphs (cf. Section 2.2.4). One interesting question, for instance, would be to find alternative characterizations of this class, and to study their trace languages (for a summary of this problem over words, see Chapter 4).

As for the rewriting graphs of suffix systems, a very interesting question concerns the decidability of the first-order logic with edge and reachability predicates over these graphs. We know from [DHLT90] that in the case of graphs defined by ground rewriting, this logic is decidable. In our state of knowledge, we miss some of the key ingredients to extend this result to suffix systems, and most importantly the closure of derivations of suffix systems under boolean operations. The following example illustrates the fact that suffix systems are strictly more general than ground systems, which is another motivation to study this family.

Example 3.26. Consider the finite suffix system $R = \{fxy \to fyx,\ a \to ga\}$ over the ranked alphabet $\{f^{(2)},\ g^{(1)},\ a^{(0)}\}$. The first rule of R allows to swap at any time both children of an f-node. This somehow expresses the commutativity of f. The derivation of R (restricted for the sake of clarity to $(fg^*ag^*a)^2$) is

$$\{(fg^mag^na, fg^nag^ma) \mid m,n \geq 0\} \cup \{(fg^mag^na, fg^{m+1}ag^na) \mid m,n \geq 0\} \cup \{(fg^mag^na, fg^mag^{n+1}a) \mid m,n \geq 0\},$$

[2]For ordinary top-down transducers, s would be a single term and each x_i a single variable.

which can not be generated by any ground tree transducer. Furthermore, we claim that the rewriting graph of this rewriting system is not isomorphic to the rewriting graph of any (recognizable) ground term rewriting system as defined in [Löd02, Col02]. Figure 3.3 illustrates the rewriting graph of R when restricted to terms in (fg^*ag^*a).

3.3.3 Verification of Parameterized Systems

Recent trends in the field of symbolic model checking of parameterized systems rely on two aspects closely related to this work on rewriting systems (see Section 2.3 for more details).

In the present case, using terms as descriptions for tree-like parameterized networks and top-down or suffix rewriting systems as models for transitions, we are able not only to ensure the existence of effective algorithms to precisely compute the system's reachability relation itself, and hence the image or pre-image of any regular set, but also guarantee that regularity of sets of configurations is preserved under reachability (only forwards in the case of top-down systems). This is very important for verification purposes, since regular sets of terms form an effective boolean algebra, which is a strong prerequisite for even the most basic symbolic verification methods.

In terms of expressiveness, such rewriting rules are able to model complex transformations modifying the structure of the network, like for instance creation and destruction of processes and modification of communication channels between them. This setting also stresses the importance of a thorough investigation of further closure and decidability properties of these systems.

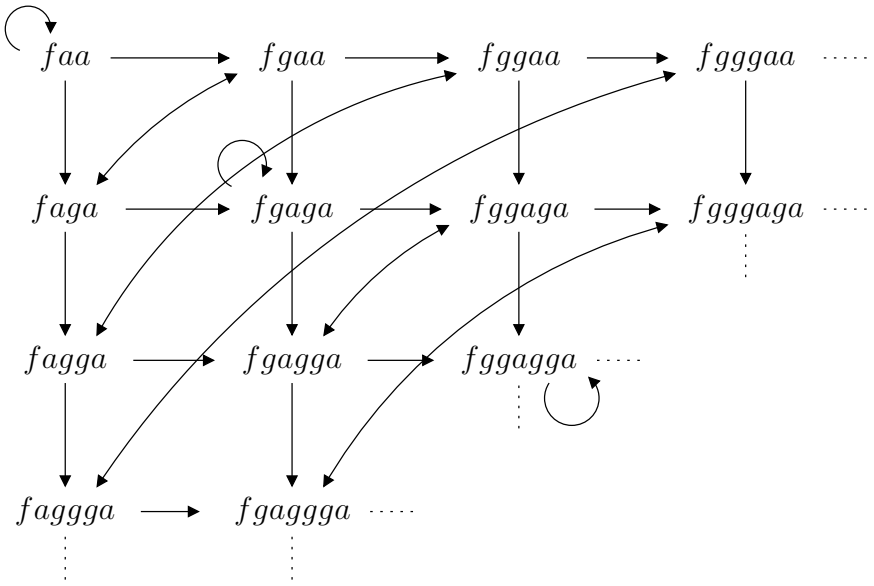


Figure 3.3: Rewriting graph of $R = \{fxy \to fyx, a \to ga\}$ over domain fg^*ag^*a.

Chapter 4

Infinite Automata for Context-Sensitive Languages

4.1 A Chomsky-like Hierarchy of Graphs

In [CK02a], the authors advertise a hierarchy of four families of infinite graphs whose families of traces coincide with the four levels of the Chomsky hierarchy. They present these families of graphs, recalled on Figure 4.1, using the homogeneous formalism of Cayley-type graphs of rewriting systems.

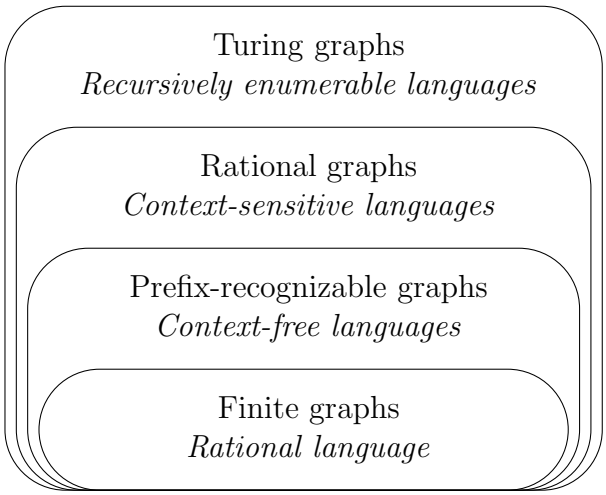


Figure 4.1: A Chomsky-like hierarchy of infinite graphs.

This chapter's concern is to further study families of graphs whose traces are the context-sensitive languages. Section 4.2 is a new formulation of existing results over rational graphs. Section 4.3 presents a new family of graphs more closely related to the usual acceptors for context-sensitive languages, namely linearly bounded Turing machines. In both cases, we will be especially concerned with the question of determinism, as well as considerations on the structure of graphs seen as infinite automata. In Section 4.4 we compare both families of graphs. We conclude this study in Section 4.5 with a proposal of another hierarchy of infinite graphs whose traces coincide with the Chomsky hierarchy, but which only considers graphs of bounded out-degree with a single initial vertex.

4.2 Rational Graphs and their Sub-Families

The first result on families of infinite graphs whose languages are the context-sensitive languages is due to Morvan and Stirling [MS01]. They showed that the languages accepted by rational graphs are precisely the context-sensitive languages. This result was later extended to the more restricted families of *synchronized* rational graphs, and even to *synchronous* graphs, by Rispal [Ris02]. All these works use context-sensitive grammars in a specific normal form defined by Penttonen [Pen74] to characterize context-sensitive languages, which has two main drawbacks. First, the Penttonen normal form is far from being obvious, and the proofs and constructions provided in [Pen74] are known to be difficult. Second, and more importantly, there is no grammar-based characterization of deterministic context-sensitive languages, which forbids one to adapt these results to the deterministic case. This particular point was addressed in earlier work by Carayol [Car01] and the author [Mey02] with the explicit concern not to use the Penttonen normal form. The aim of this section is to give a uniform and enhanced presentation of these new constructions.

In a first time, we consider the general case where the set of initial vertices can be given by any rational set of words. In this setting, we re-establish and extend previous results without using grammars. Our approach is based on tiling systems, which are finite acceptors for context-sensitive languages introduced in [LS97]. As shown in appendix A, tiling systems are, contrary to grammars in Penttonen normal form, syntactically equivalent to linearly bounded Turing machines (LBMs). In particular, they admit a natural notion of determinism which coincides with the determinism of LBMs. After showing that any rational graph is trace-equivalent to a rational graph with synchronous (or length-preserving) transducers, we prove the tight relationship between synchronous rational graphs and tiling systems. Moreover, we show that it is possible to accept any context-sensitive language with synchronous transducers that deterministically consume

their input (also called sequential synchronous transducers).

As synchronized graphs, unlike rational graphs, enjoy a decidable first-order theory [BG00], these results could lead to believe that they are the most suitable notion of infinite acceptors for context-sensitive languages. However, when considering simple structural restrictions (unique initial vertex and finite out-degree), the rational graphs still accept all context-sensitive languages while synchronized rational graphs only accept the languages recognized by LBMs working with a linear number of head reversals, which can be reasonably assumed to be a strict sub-family of context-sensitive languages. We also provide some partial results concerning the classes of languages obtained when bounding the out-degree. Finally, we show how these new constructions can be used to approach satisfactory notions of determinism for infinite graphs seen as automata: we define a syntactical sub-family of rational graphs accepting precisely the deterministic context-sensitive languages.

This section is structured along the following lines. Necessary additional definitions concerning context-sensitive languages are given in Section 4.2.1. The results concerning languages accepted by rational graphs and their sub-families appear in Section 4.2.2. In Section 4.2.3, we investigate rational graphs under various structural constraints. Finally, Section 4.2.4 investigates the relation between rational graphs and deterministic context-sensitive languages.

Some of the results we present are summarized in Table 4.1. An equality symbol indicates that the languages accepted by the considered family of graphs (row) from the considered set of initial vertices (column) are the context-sensitive languages. An inclusion symbol indicates that their languages are strictly included in context-sensitive languages. A question mark denotes a conjecture. Whenever necessary, a reference to the relevant proposition, theorem or remark in the paper is given. Redundant results or corollaries are omitted.

	Rational set	Set i^*	One vertex	One vertex (finite deg.)
Rational [MS01]	$=$	$=$	$=$	$=$ [4.28]
Synchronized [Ris02]	$=$	$=$	$=$ [4.23]	$\subset$ (?) [4.32]
Synchronous [Ris02]	$=$	$=$ [4.17]	$\subset$ [4.24]	$\subset$
Sequ. synchronous	$=$ [4.20]	$\subset$ (?) [4.40]	$\subset$	$\subset$

Table 4.1: Families of rational graphs and their languages.

4.2.1 Other Acceptors for Context-Sensitive Languages

This section presents two alternative families of acceptors for context-sensitive languages, namely tiling systems and cellular automata with borders. We give the definition of each formalism and state an equivalence property of both families of acceptors with linearly bounded machines. In this chapter, in order to simplify our presentation, we only consider context-sensitive languages that do not contain the empty word ε (this is a standard restriction).

4.2.1.1 Tiling systems

A not-so-well-known formalism accepting context-sensitive languages are *tiling systems* with borders. Tiling systems were originally defined to recognize or specify *picture languages*, i.e. two-dimensional words on finite alphabets [GR96]. Such sets of pictures are called *local* picture languages. However, by only looking at the words contained in the first row of each picture of a local picture language, one gets a context-sensitive language. Conversely, every context-sensitive language can be seen as the set of first rows of a local picture language.

A (n, m)-picture p over an alphabet Γ is a two dimensional array of letters in Γ with n rows and m columns. We denote by $p(i, j)$ the letter occurring in the ith row and jth column starting from the top-left corner. We denote by $\Gamma^{n,m}$ the set of (n, m)-pictures and by Γ^{**} the set of all pictures[1].

Given a (n, m)-picture p over Γ and a letter $\# \notin \Gamma$, we denote by $p_\#$ the $(n + 2, m + 2)$-picture over $\Gamma \cup \{\#\}$ defined by:

- $p_\#(i, 1) = p_\#(i, m + 2) = \#$ for $i \in [1, n + 2]$,
- $p_\#(1, j) = p_\#(n + 2, j) = \#$ for $j \in [1, m + 2]$,
- $p_\#(i + 1, j + 1) = p(i, j)$ for $i \in [1, n]$ and $j \in [1, m]$.

For any $n, m \geq 2$ and any (n, m)-picture p, $T(p)$ is the set of $(2, 2)$-pictures appearing in p. A $(2, 2)$-picture is also called a *tile*. A picture language $K \subseteq \Gamma^{**}$ is *local* if there exists a symbol $\# \notin \Gamma$ and a finite set of tiles Δ such that $K = \{p \in \Gamma^{**} \mid T(p_\#) \subseteq \Delta\}$. To any set of pictures over Γ, we can associate a language of words by looking at the frontiers of the pictures. The frontier of a (n, m)-picture p is the word $\mathrm{fr}\,(p) = p(1, 1) \dots p(1, m)$ corresponding to the first row of the picture.

Definition 4.1. A tiling system S is a tuple $(\Gamma, \Sigma, \#, \Delta)$ where Γ is a finite alphabet, $\Sigma \subset \Gamma$ is the input alphabet, $\# \notin \Gamma$ is a frame symbol and Δ is a finite set of tiles over $\Gamma \cup \{\#\}$. It recognizes the local picture language $P(S) = \{p \in \Gamma^{**} \mid T(p_\#) \subseteq \Delta\}$ and the associated language $L(S) = \mathrm{fr}\,(P(S)) \cap \Sigma^*$.

[1] We do not consider the empty picture.

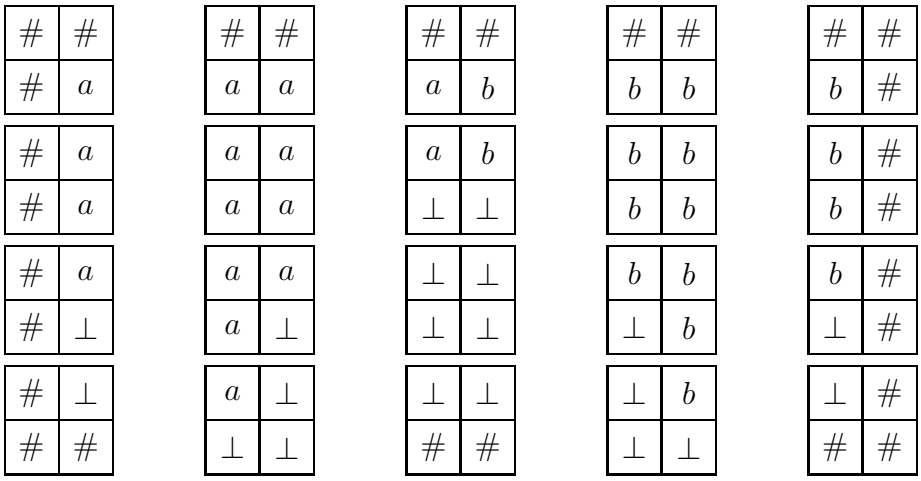


Figure 4.2: A tiling system accepting $\{a^n b^n \mid n \geq 1\}$ (Cf. Example 4.3).

A tiling system S recognizes a language $L \subseteq \Sigma^+$ in height f for some mapping $f : \mathbb{N} \mapsto \mathbb{N}$ if for all $w \in L(S)$ there exists a (n, m)-picture p in $P(S)$ such that $w = \mathrm{fr}\,(p)$ and $n \leq f(m)$.

There is no standard notion of determinism for tiling systems used as language recognizers. However, it is possible to characterize a class of tiling systems whose languages are the deterministic context-sensitive languages. We will call them *deterministic*, by lack of a better word.

Definition 4.2. A tiling system $S = (\Gamma, \Sigma, \#, \Delta)$ is deterministic if, for all word $u \in \#\Gamma^*\#$, there is at most one word $v \in \#\Gamma^*\#$ such that $|v| = |u|$ and that the picture p with rows u and v has tiles in Δ.

This means that one can deterministically infer each row in a picture from the previous one.

Example 4.3. Figure 4.2 shows the set of tiles Δ of a tiling system S over $\Gamma = \{a, b, \bot\}$, $\Sigma = \{a, b\}$ and the border symbol $\#$. The language $L(S)$ is exactly the set $\{a^n b^n \mid n \geq 1\}$. Figure 4.3 shows an element p of $P(S)$ and the corresponding framed picture $p_\#$. Note that in this case, $T(p_\#) = \Delta$. Also note that this tiling system is deterministic.

4.2.1.2 Cellular automata

A third family of models which recognize context-sensitive languages, apart from LBMs and tiling systems, are cellular automata working on a fixed space. As for

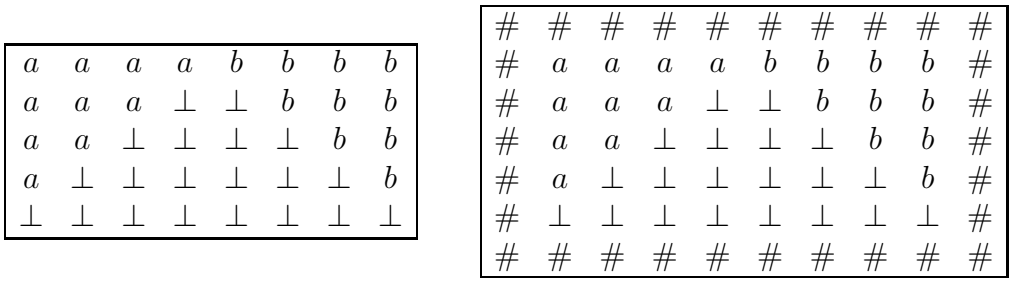


Figure 4.3: A picture accepted by the set of tiles of Figure 4.2.

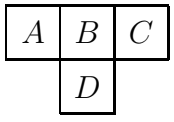
---

tiling systems, they are not traditionally studied as language recognizers. However, one can see a cellular automaton as a machine operating on configurations. A word is accepted if, in a finite number of steps, the automaton transforms it into a certain accepting configuration. In contrast with the traditional approach [MD98], we consider cellular automata which can be non-deterministic.

Definition 4.4. A *cellular automaton* C is a tuple $(\Gamma, \Sigma, F, [,], \delta)$ where:

- Γ and $\Sigma \subset \Gamma$ are respectively the work and input alphabet,
- $F \subset \Gamma$ is the set of terminal symbols,
- [and] are symbols which do not belong to Γ,
- $\delta \subseteq (\Gamma \cup \{ [\}) \times \Gamma \times (\Gamma \cup \{] \}) \times \Gamma$ is the transition function.

We will use the following graphical representation for a transition (A, B, C, D):



Quite naturally, we call a cellular automaton *deterministic* if, for every triple (A, B, C), there exists at most one D such that $(A, B, C, D) \in \delta$.

A configuration[2] is a word of the form $[u]$ where $u \in \Gamma^+$. The cellular automaton C defines a successor relation on configurations: $c' = [v]$ is a successor of $c = [u]$ if $|c| = |c'| = n$ and for all $i \in [2, n-1]$, $(c(i-1), c(i), c(i+1), c'(i)) \in \delta$. An *initial configuration* is a configuration $[w]$ where $w \in \Sigma^+$ and a *final configuration* is a configuration of the form $[F^+]$. The notions of run or computation, acceptance in n steps and recognized language are defined as for linearly bounded Turing machines.

[2]Remember that we only consider languages that do not contain the empty word.

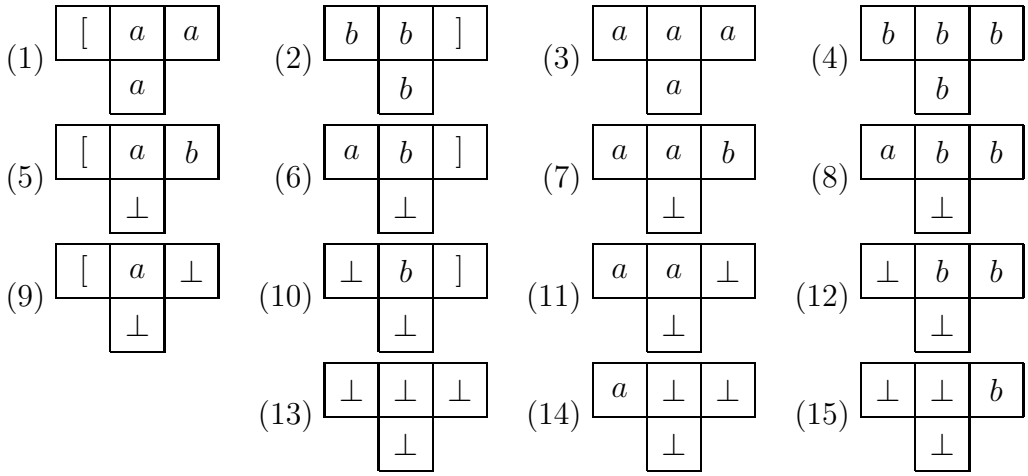


Figure 4.4: A cellular automaton accepting $\{a^n b^n \mid n \geq 1\}$ (Cf. Example 4.5).

Example 4.5. Figure 4.4 shows the set of rules δ of a deterministic cellular automaton $C = (\Gamma, \Sigma, \{\bot\}, [,], \delta)$ which can transform configurations of the form $[a^n \bot^k b^m]$ with $m, n > 0$ (and only those configurations) into $[a^{n-1} \bot^{k+2} b^{m-1}]$. This cellular automaton can only reach an accepting configuration if $m = n$, hence it accepts the language $\{a^n b^n \mid n \geq 1\}$.

4.2.1.3 Relations between the acceptors

In [Kur64], Kuroda showed that the languages accepted by linearly bounded Turing machines are the same as the languages generated by growing grammars, namely context-sensitive languages. Latteux and Simplot proved in [LS97] that it was also the case for tiling systems seen as language recognizers[3]. Similar results are part of the folklore for cellular automata. To convince the reader that these acceptors are in fact syntactically equivalent, self-contained proofs are given in Appendix A.

Theorem 4.6. *The following simulations link linearly bounded Turing machines, cellular automata and tiling systems:*

1. *A linearly bounded Turing machine T working in $f(n)$ reversals can be simulated by a cellular automaton C working in time $f(n) + 2$.*

[3]The construction given by Latteux and Simplot has a higher complexity bound than the one stated in Theorem 4.6. The corresponding construction is given in appendix (see Proposition A.4).

2. *A cellular automaton C working in time $f(n)$ can be simulated by a tiling system S of height $f(n)$.*
3. *A tiling system of height $f(n)$ can be simulated by a linearly bounded Turing machine T working in $f(n)$ reversals.*

This result is not surprising, given the very close resemblance between all these formalisms. Interestingly, a deterministic version of this theorem can also be established with respect to the definitions of determinism for the various acceptors given previously.

Theorem 4.7. *For any language L, the following facts are equivalent:*

1. *L is a deterministic context-sensitive language,*
2. *L is recognized by a deterministic linearly bounded Turing machine,*
3. *L is recognized by a deterministic cellular automaton,*
4. *L is recognized by a deterministic tiling system.*

From this point on, we will be able to always choose the type of context-sensitive acceptors which suits us best in a given situation.

4.2.2 The languages of rational graphs

In this section, we consider the languages accepted by rational graphs and their sub-families from and to a rational set of vertices. We give a simplified presentation of the result by Morvan and Stirling [MS01] stating that the family of rational graphs accepts the context-sensitive languages. This is done in several steps. First, Proposition 4.10 states that the rational graphs are trace-equivalent to the *synchronous* rational graphs. Then, Proposition 4.11 and Proposition 4.15 establish a very tight relationship between synchronous graphs and tiling systems. It follows that the languages of synchronous rational graphs are also the context-sensitive languages (Theorem 4.17). The original result is given as Corollary 4.18. Finally, Proposition 4.20 establishes that even the smallest sub-family we consider, the family of sequential synchronous rational graphs, accepts all context-sensitive languages. The various transformations presented in this section are summarized in Figure 4.5.

4.2.2.1 From rational graphs to synchronous graphs

We present an effective construction that transforms a rational graph G with two rational sets I and F of initial and final vertices into a trace-equivalent synchronous graph G' between two rational sets I' and F'. The construction is based on replacing the symbol ε in the transitions of the transducers defining G by a fresh symbol #.

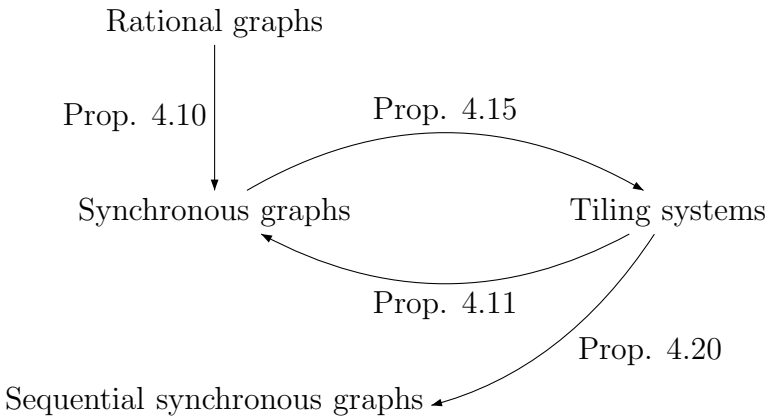


Figure 4.5: Each edge represents an effective transformation preserving languages.

Let $(T_a)_{a\in\Sigma}$ be the set of transducers over Γ characterizing G and let $\#$ be a symbol not in Γ. For all a, we define $\bar{a}$ to be equal to a if $a \in \Gamma$, and to ε if $a = \#$. We extend this to a projection from $(\Gamma \cup \#)^*$ to Γ^* in the standard way. We define G' as the rational graph defined by the set of transducers $(T'_a)_{a\in\Sigma}$ where T'_a has the same set of control states Q_a as T_a and a set of transitions given by

$$\left\{ p \xrightarrow{a/b} q \mid p \xrightarrow{\bar{a}/\bar{b}} q \in T_a \right\} \cup \left\{ p \xrightarrow{\#/\#} p \mid p \in Q_a \right\}.$$

By definition of each T'_a, G' is a synchronous rational graph. Let I' and F' be the two rational sets such that $I' = \{u \mid \bar{u} \in I\}$ and $F' = \{v \mid \bar{v} \in F\}$ (the automaton accepting I' (resp. F') is obtained from the automaton accepting I (resp. F) by adding a loop labeled by $\#$ on each control state). We claim that G' accepts between I' and F' the same language as G between I and F. For example, Figure 4.6 illustrates the previous construction applied to the graph of Figure 2.6. Only one connected component of the obtained graph is shown.

Before we prove the correctness of this construction, we need to establish a couple of technical lemmas. Let $\mathcal{B}$ be the set of all mappings from $\mathbb{N}$ to $\mathbb{N}$. To any mapping $\delta \in \mathcal{B}$, we associate a mapping from $(\Gamma \cup \{\#\})^*$ to $(\Gamma \cup \{\#\})^*$ defined as follows: for all $w = \#^{i_0}a_1\#^{i_1} \dots a_n\#^{i_n}$ with $a_1, \dots, a_n \in \Gamma$, let $\delta w = \#^{i_0+\delta(0)}a_1\#^{i_1+\delta(1)} \dots a_n\#^{i_n+\delta(n)}$. Before proceeding, we state two properties of these mappings with respect to the sets of transducers (T_a) and (T'_a).

Lemma 4.8. $\forall u, v \in \Gamma^*,\ (u, v) \in T_a \iff \exists \delta_u, \delta_v \in \mathcal{B},\ (\delta_u u, \delta_v v) \in T'_a$.

Proof. We first show that for all path ρ between states p and q in T_a labeled by u/v, there is a pair of mappings δ_u, δ_v such that there is a path in T'_a labeled by

$\delta_u u/\delta_v v$ between p and q. This is done by induction on the length of ρ. If $|\rho| = 0$, one can simply take $\delta_u(i) = \delta_v(i) = 0$ for all i. Otherwise, if $\rho = k > 0$, there is some state p' such that ρ can be written $p \xrightarrow{x/y} p' \xrightarrow{u'/v'} q$ with $x, y \in \Gamma \cup \{\varepsilon\}$ and $u', v' \in \Gamma^*$. By induction hypothesis, one can find mappings $\delta_{u'}$ and $\delta_{v'}$ such that $\delta_{u'}u'/\delta_{v'}v'$ labels a path between p' and q in T'_a. By construction, there exists a transition $p \xrightarrow{x'/y'} p'$ in T'_a such that $x = \bar{x}'$ and $y = \bar{y}'$. Hence for

$$\delta_u = \begin{cases} \{0 \mapsto 0\} \cup \{i \mapsto \delta_{u'}(i-1) \mid i > 0\} & \text{if } x \neq \varepsilon \\ \{0 \mapsto \delta_{u'}(0) + 1\} \cup \{i \mapsto \delta_{u'}(i) \mid i > 0\} & \text{if } x = \varepsilon \end{cases}$$

and δ_v defined similarly with respect to $\delta_{v'}$ and y, we have $p \xrightarrow{\delta_u u/\delta_v v} q \in T'_a$.

Conversely, if there is a path in T'_a labeled by u'/v', then by construction there must also be a path in T_a labeled by $\bar{u}'/\bar{v}'$. Hence for any pair of mappings δ_u, δ_v such that $\delta_u \bar{u}' = u'$ and $\delta_v \bar{v}' = v'$, the property is verified. □

We now state a second property of the set of mappings $\mathcal{B}$.

Lemma 4.9. $\forall (u, v) \in T'_a,\ \forall \delta \in \mathcal{B},\ \exists \delta' \in \mathcal{B},\ (\delta u, \delta' v) \in T'_a$,
and dually $\forall (u, v) \in T'_a,\ \forall \delta \in \mathcal{B},\ \exists \delta' \in \mathcal{B},\ (\delta' u, \delta v) \in T'_a$.

Proof. We will prove a slightly more general property than the one expressed by the lemma. Consider any path ρ between two control states p and q labeled by a pair of words (u, v) in transducer T'_a, and any mapping $\delta \in \mathcal{B}$. Our aim is to define another mapping δ' such that there is a path between p and q in T'_a labeled by $(\delta u, \delta' v)$. This new path will be obtained from the original path in T_a by adding loops labeled by $\#/\#$. The existence of δ' can be proved by induction on the size of ρ.
Base case. As T'_a is synchronous and $|\rho| = 0$, we have $u = v = \varepsilon$ and $p = q$. If we take $\delta'(0) = \delta(0)$ and $\delta'(i) = 0$ for all $i > 0$, then $\delta u = \delta' v = \#^{\delta(0)}$. This path can be obtained in T'_a by taking $\delta(0)$ times the $\#/\#$ loop on p.
Inductive case. Suppose the property is true for all paths of length less than or equal to n, and let ρ be a path of length n in T'_a labeled by u/v between states p and q. By induction hypothesis, for all δ there is a mapping δ' such that $\delta u/\delta' v$ also labels a path between p and q. Now consider adding at the end of ρ a transition labeled by x/y between q and some state q'. We will define a new mapping δ'' from δ' according to the values of x and y. There are four cases to consider:

1. If $x = \#,\ y = \#, \quad \delta'' = \delta'$
2. If $x \in \Gamma,\ y \in \Gamma, \quad \delta''(i) = \begin{cases} \delta'(i) & \text{if } i \leq |\bar{v}| \\ \delta(|\bar{u}| + 1) & \text{if } i = |\bar{v}| + 1 \end{cases}$

3. If $x \in \Gamma,\ y = \#, \quad \delta''(i) = \begin{cases} \delta'(i) & \text{if } i < |\bar{v}| \\ \delta'(i) + \delta(|\bar{u}| + 1) & \text{if } i = |\bar{v}| \end{cases}$

4. If $x = \#,\ y \in \Gamma, \quad \delta''(i) = \begin{cases} \delta'(i) & \text{if } i \leq |\bar{v}| \\ 0 & \text{if } i = |\bar{v}| + 1 \end{cases}$

When not precised, the value of $\delta''(i)$ is assumed to be 0. We claim that this definition ensures that $\delta ux/\delta'' vy$ labels a valid path in T'_a between p and q', which concludes the proof by induction. In the case where ρ is accepting, we get the proof of the first property. The dual is proved in the same way. □

We can now prove the correctness of the construction: a word w is accepted by G between I and F if and only if it is accepted by G' between I' and F'.

Proposition 4.10. *For all rational graph G and rational sets of vertices I and F, there is a synchronous rational graph G' and two rational sets I' and F' such that $L(G, I, F) = L(G', I', F')$.*

Proof. We show by induction on n that for all $u_0, \ldots, u_n \in \Gamma^*$, if there is a path

$$u_0 \xrightarrow[G]{w(1)} u_1 \ldots u_{n-1} \xrightarrow[G]{w(n)} u_n,$$

then there exist words $u'_0, \ldots, u'_n \in (\Gamma \cup \{\#\})^*$ such that for all i, $\bar{u}'_i = u_i$, and

$$u'_0 \xrightarrow[G']{w(1)} u'_1 \ldots u'_{n-1} \xrightarrow[G']{w(n)} u'_n.$$

The case where $n = 0$ is trivial. Suppose the property is true for all paths of length at most n, and consider a path

$$u_0 \xrightarrow[G]{w(1)} \ldots \xrightarrow[G]{w(n)} u_n \xrightarrow[G]{w(n+1)} u_{n+1}.$$

By induction hypothesis, one can find mappings $\delta_0, \ldots \delta_n$ such that

$$\delta_0 u_0 \xrightarrow[G']{w(1)} \ldots \xrightarrow[G']{w(n)} \delta_n u_n.$$

By Lemma 4.8, there exist δ'_n and δ'_{n+1} such that $\delta'_n u_n \xrightarrow[G']{w(n+1)} \delta'_{n+1} u_{n+1}$. Let γ_n and γ'_n be two elements of $\mathcal{B}$ such that $\delta'_n \circ \gamma'_n = \delta_n \circ \gamma_n$. By Lemma 4.9, we can find mappings γ'_{n+1} and γ_0 to γ_{n-1} such that:

$$\gamma_0 \delta_0 u_0 \xrightarrow[G']{w(1)} \ldots \xrightarrow[G']{w(n)} \gamma_n \delta_n u_n = \gamma'_n \delta'_n u_n \xrightarrow[G']{w(n+1)} \gamma'_{n+1} \delta'_{n+1} u_{n+1}$$

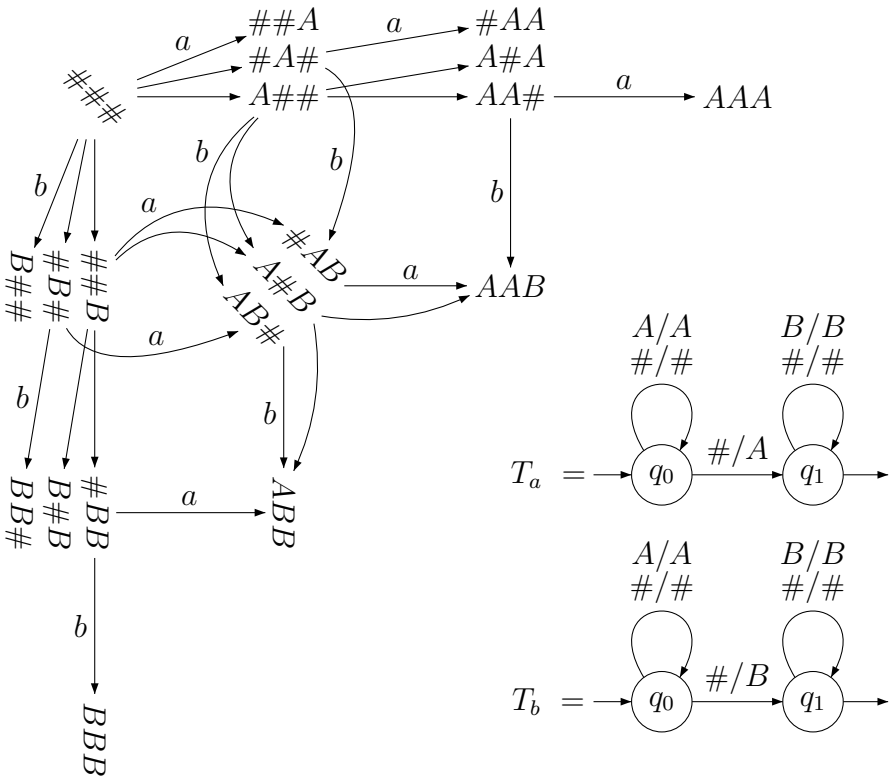


Figure 4.6: Synchronous graph trace-equivalent to the grid (1 connected component).

---

which concludes the proof by induction. If we suppose that $u_0 \in I$ and $u_n \in F$, then necessarily $u'_0 \in I'$ and $u'_n \in F'$. It follows that for all path in G between I and F, there is a path in G' between I' and F' with the same path label.

Conversely, by Lemma (4.8), for any such path in G', erasing the occurrences of $\#$ from its vertices yields a valid path in G between I and F. Hence $L(G, I, F) = L(G', I', F')$. □

4.2.2.2 Equivalence of synchronized graphs and tiling systems

The following propositions establish the tight relationship between tiling systems and synchronous rational graphs. Proposition 4.11 presents an effective transformation of a tiling system into a synchronous rational graph.

Proposition 4.11. *Given a tiling system $S = (\Gamma, \Sigma, \#, \Delta)$, there exists a synchronous rational graph G and two rational sets I and F such that $L(G, I, F) = L(S)$.*

Proof. Consider the finite automaton A on Γ with a set of states $Q = \Gamma \cup \{\#\}$, initial state $\#$, a set of final states F and a set of transitions δ given by:

$$F \ : \ a \quad \text{such that} \quad \begin{array}{|c|c|}\hline a & \# \\ \hline \# & \# \\ \hline \end{array} \in \Delta$$

$$\delta \ : \ \# \xrightarrow[A]{a} a, \ a \xrightarrow[A]{b} b \quad \text{for all} \quad \begin{array}{|c|c|}\hline \# & \# \\ \hline a & \# \\ \hline \end{array}, \ \begin{array}{|c|c|}\hline a & \# \\ \hline b & \# \\ \hline \end{array} \in \Delta \text{ (respectively).}$$

Call M the language recognized by A, M represents the set of possible last columns of pictures of $P(S)$. Note that this does not imply that each word of M actually *is* the last column of a picture in $P(S)$, only that it is compatible with the right border tiles of Δ.

Let us build a synchronous rational graph G and two rational sets I and F such that $L(G, I, F) = L(S)$. The transitions of the set of transducers $(T_e)_{e\in\Sigma}$ of G are:

$$(\#, \#) \xrightarrow[T_d]{c/d} (c, d) \qquad \text{for all} \quad \begin{array}{|c|c|}\hline \# & \# \\ \hline c & d \\ \hline \end{array} \in \Delta, \ d \neq \#$$

$$(a, b) \xrightarrow[T_e]{c/d} (c, d) \qquad \text{for all} \quad \begin{array}{|c|c|}\hline a & b \\ \hline c & d \\ \hline \end{array} \in \Delta, \ b, d \neq \#, \ e \in \Sigma$$

where $(\#, \#)$ is the unique initial state of each transducer and the set of final states F of each transducer is given by:

$$F \ : \ (a, b) \in (\Gamma \cup \{\#\}) \times \Gamma \ \text{ such that } \ \begin{array}{|c|c|}\hline a & b \\ \hline \# & \# \\ \hline \end{array} \in \Delta.$$

A pair of words (s, t) is accepted by the transducer T_e if and only if e is the first letter of t, and either s and t are two adjacent columns of a picture in $P(S)$ or $s \in \#^*$ and t is the first column of a picture in $P(S)$. As a consequence, $L(S) = L(G, \#^*, M)$. □

Example 4.12. Figure 4.7 shows the transducers obtained using the previous construction on the tiling system of Figure 4.2. They define a rational graph whose path language between $\#^*$ and $b^*\perp$ is $\{a^n b^n \mid n \geq 1\}$. Figure 4.8 presents the corresponding synchronous graph whose vertices are the rational set of words $\#^{\geq 2} \ \cup \ a^+\perp^+ \ \cup \ b^+\perp^+$, the set of initial vertices is $\#^{\geq 2}$ and the set of final vertices is $b^+\perp$. Remark that in this example, the set of vertices accessible from the initial vertices is rational: this is not true in the general case.

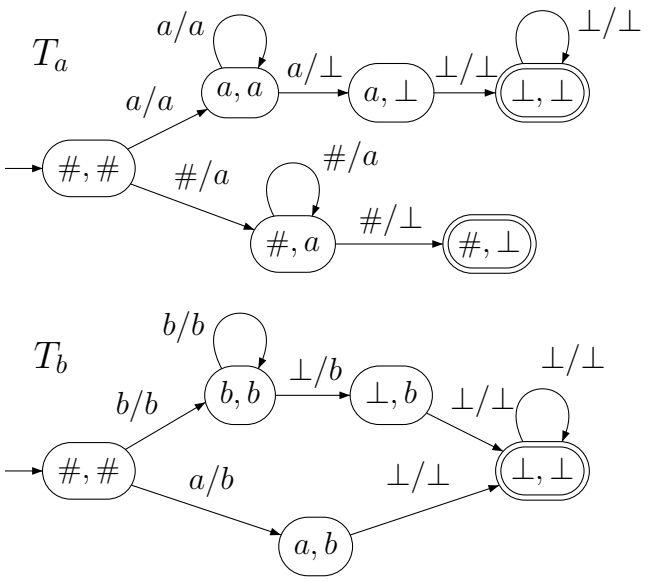


Figure 4.7: Transducers of a synchronous graph accepting $\{a^n b^n \mid n \geq 1\}$.

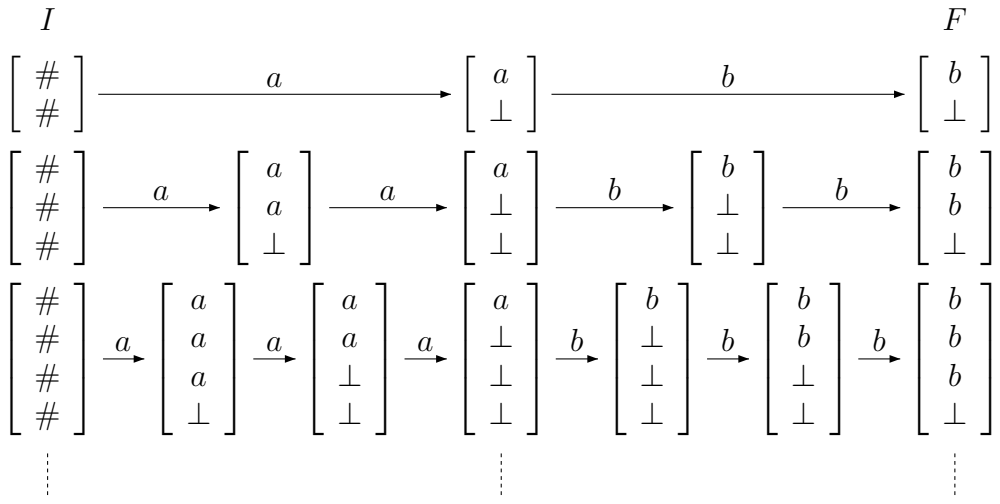


Figure 4.8: The synchronous rational graph associated to the tiling system of Figure 4.2 whose transducers are presented in Figure 4.7.

Remark 4.13. The correspondence between a tiling system S and the synchronous graph G constructed from S in Proposition 4.11 is tight: each picture p with frontier w corresponds to a unique accepting path for w in G (and conversely). More formally, there exists a bijection ϕ between the pictures in $P(S)$ and the accepting path in G from I to F. In particular, for all $p \in P(S)$ with frontier w and of height n, $\phi(p)$ is an accepting path for w with vertices of length n.

To make the proof of the converse construction simpler, we first prove that the sets of initial and final vertices can be chosen over a one-letter alphabet without loss of generality.

Lemma 4.14. *For all synchronous rational graph G with vertices in Γ^* and rational sets I and F, one can find a synchronous rational graph H and two symbols i and $f \notin \Gamma$ such that $L(G, I, F) = L(H, i^*, f^*)$.*

Proof. Let $G = (K_a)_{a\in\Sigma}$ be a synchronous rational graph with vertices in Γ^*. For i, f two new distinct symbols, we define a new synchronous rational graph H characterized by the set of transductions

$$\Big(T_a = (T_I \circ K_a) \ \cup \ K_a \ \cup \ (K_a \circ T_F)\Big)_{a\in\Sigma}$$

where $T_I = \{(i^n, u) \mid n \geq 0,\ u \in I,\ |u| = n\}$ and $T_F = \{(v, f^n) \mid n \geq 0,\ v \in F,\ |v| = n\}$. For all vertices $u \in I,\ v \in F$ we have $u \xrightarrow[G]{w} v$ if and only if $i^{|u|} \xrightarrow[H]{w} f^{|u|}$, i.e. $L(G, I, F) = L(H, i^*, f^*)$. □

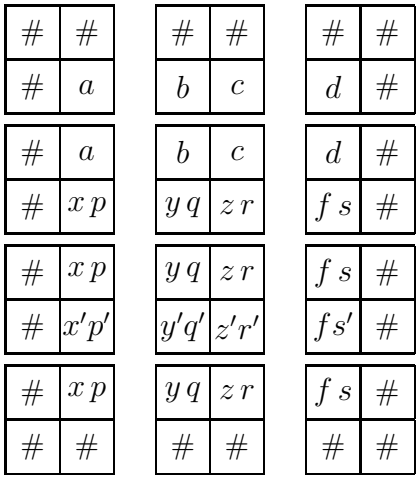
We are now able to establish the converse of Proposition 4.11, which states that all the languages accepted by synchronous rational graphs between rational sets of vertices can be accepted by a tiling system.

Proposition 4.15. *Given a synchronous rational graph G and two rational sets I and F, there exists a tiling system S such that $L(S) = L(G, I, F)$.*

Proof. Let $G = (T_a)_{a\in\Sigma}$ be a synchronous rational graph with vertices in Γ^* (with $\Sigma \subseteq \Gamma$). By Lemma 4.14, we can consider without loss of generality that $I = i^*$ and $F = f^*$ for some distinct letters i and f, and that neither i nor f occurs in any vertex which is not in I or F. Furthermore by Remark 1.28, we can assume that T_a is non-ambiguous for all $a \in \Sigma$.

We write Q_a the set of control states of T_a. We suppose that all control state sets are disjoint, and designate by $q_0^a \in Q_a$ the unique initial state of each transducer T_a, and by Q_F the set of final states of all T_a.

Let $a, b, c, d \in \Sigma$, $x, x', y, y', z, z' \in \Gamma$, and $p, p', q, q', r, r', s, s' \in \bigcup_{a\in\Sigma} Q_a$. We define a tiling system $S = (\Gamma, \Sigma, \#, \Delta)$, where Δ is the following set of tiles:



with $q_0^a \xrightarrow[T_a]{i/x} p$, $q_0^c \xrightarrow[T_c]{y/z} r$, $s \in Q_d$

with $\exists a, b, c,\ p \xrightarrow[T_a]{i/x'} p',\ r \xrightarrow[T_b]{y'/z'} r',\ s, s' \in Q_c$

with $p, q, r, s \in Q_F$

By construction, $P(S)$ is in exact bijection with the set of accepting paths in G with respect to I and F. Let ϕ be the function associating to a picture $p \in P(S)$ with columns $a_1w_1, \ldots, a_nw_n$, the path $i^{|w_1|} \xrightarrow{a_1} \widetilde{w_1} \ldots \xrightarrow{a_n} \widetilde{w_n}$ where $\widetilde{w}$ is obtained by removing the control states from w. By construction of S, the function ϕ is well defined. It is easy to check that ϕ is an onto function. As the transducers defining G are non-ambiguous, two distinct pictures have distinct images by ϕ and therefore ϕ is an injection.

Hence, the tiling system $(\Gamma, \Sigma, \#, \Delta)$ exactly recognizes $L(G, I, F)$. □

Remark 4.16. As in Remark 4.13, the set of paths in G from I to F and the set of pictures $P(S)$ accepted by S are in bijection, and the length of the vertices along the path is equal to the height of the corresponding picture.

Putting together Propositions 4.11 and 4.15 and Theorem 4.6, we obtain the following result concerning the path languages of synchronous rational graphs.

Theorem 4.17 ([Ris02])**.** *The languages accepted by the synchronous rational graphs between rational sets of initial and final vertices are the context-sensitive languages.*

Note that this formulation of the theorem could be made a bit more precise by recalling that initial and final sets of vertices only of the form x^*, where x is a letter, are sufficient to accept all context-sensitive languages, as stated in Lemma 4.14. By Proposition 4.10, this implies as a corollary the original result by Morvan and Stirling [MS01].

Corollary 4.18. *The languages accepted by rational graphs between rational sets of initial and final vertices are the context-sensitive languages.*

This result can be slightly strengthened in the case of a single initial and final vertex.

Corollary 4.19. *For any rational graph G labeled by Σ and any symbol $\sharp \notin \Sigma$, the language $L_G = \{i\sharp w\sharp f \mid w \in L(G, i, f)\}$ is context-sensitive.*

Proof. Let G be a rational graph labeled by Σ with vertices in Γ^* and defined by a family of transducers $(T_a)_{a\in\Sigma}$. Let $\bar{\Gamma}$ and $\tilde{\Gamma}$ be two finite alphabets disjoint from but in bijection with Γ. For any $x \in \Gamma$, we write $\bar{x}$ (resp. $\tilde{x}$) the corresponding symbol in $\bar{\Gamma}$ (resp. $\tilde{\Gamma}$). We consider the rational graph H labeled by $\Xi = \Sigma \cup \bar{\Gamma} \cup \tilde{\Gamma}$ defined by the family of transducers $(T_x)_{x\in\Xi}$ where for all $x \in \Gamma$, $T_{\bar{x}} = \{(u, ux) \mid u \in \Gamma^*\}$ and $T_{\tilde{x}} = \{(xu, u) \mid u \in \Gamma^*\}$.

By Corollary 4.18, the language $L = L(H, \varepsilon, \varepsilon) \cap \bar{\Gamma}^*\Sigma^*\tilde{\Gamma}^*$ is a context-sensitive language. By construction, L is equal to $\{\bar{i}w\tilde{f} \mid i, f \in \Gamma^* \text{ and } w \in \Sigma^*\}$. It follows that L_G is context-sensitive. □

Note that if we transform a rational graph into a Turing machine by successively applying the constructions of Proposition 4.10, Proposition 4.15 and Theorem 4.7, we obtain the same Turing machine as in [MS01].

4.2.2.3 Sequential synchronous graphs are enough

Theorem 4.17 shows that when considering rational sets of initial and final vertices, synchronous graphs are enough to accept all context-sensitive languages. It is natural to wonder whether this still holds for rational graphs defined by even more restricted families of transducers. Interestingly, when considering rational sets of initial and final vertices, the very poor class of sequential synchronous transducers are sufficient, as stated by the following proposition.

Proposition 4.20. *The languages accepted by sequential synchronous rational graphs between rational sets of initial and final vertices are the context-sensitive languages.*

Proof. Thanks to Proposition 4.15, it suffices to prove that any context sensitive language $L \subseteq \Sigma^*$ is accepted by a synchronous sequential rational graph. By Theorem 4.6, we know that there exists a tiling system $S = (\Gamma, \Sigma, \#, \Delta)$ such that $L(S) = L$.

Let $\Lambda = \Gamma \cup \{\#\}$ and [and] be two symbols that do not belong to Λ. We associate to each picture $p \in \Lambda^{**}$ with rows $l_1, \ldots, l_n$ the word $[l_1] \ldots [l_n]$. We are going to define a set of sequential synchronous transducers that, when iterated, recognize the words corresponding to pictures in $P(S)$.

First, for any finite set of tiles Δ, we construct a transducer T_Δ which checks that a word in $([\Lambda^{\geq 3}])^{\geq 2}$ represents a picture with tiles in Δ. The checking is

done column by column, and we introduce marked letters to keep track of the column being checked. Let $\widetilde{\Lambda}$ be a finite alphabet in bijection with but disjoint from Λ. For all $x \in \Lambda$ we write $\widetilde{x} \in \widetilde{\Lambda}$ the marked version of x. For all word $w = u\widetilde{x}v \in \Lambda^*\widetilde{\Lambda}\Lambda^*$, we write $\pi(w)$ the word $uxv \in \Lambda^*$ and $\rho(w) = |u| + 1$ designates the position of the marked letter in the word.

We consider words in $[\Lambda^*\widetilde{\Lambda}\Lambda^*]^{\geq 2}$. Let Shift be the relation that shifts all marks in a word one letter to the right. More precisely, Shift satisfies $Dom(\text{Shift}) = ([\Lambda^*\widetilde{\Lambda}\Lambda^+])^{\geq 2}$, and $\text{Shift}([w_1]\dots[w_n]) = [w'_1]\dots[w'_n]$ with $\pi(w'_i) = \pi(w_i)$ and $\rho(w'_i) = \rho(w_i) + 1$ for all $i \in [1, n]$. The rational relation Shift can be realized by a synchronous sequential transducer T_{Sh}. Consider the following rational language:

$$R_\Delta = \left\{ [w_1x_1\widetilde{y_1}w'_1]\dots[w_nx_n\widetilde{y_n}w'_n] \mid n \geq 2 \text{ and } \forall i \in [2,n], \begin{array}{|c|c|}\hline x_{i-1} & y_{i-1} \\ \hline x_i & y_i \\ \hline \end{array} \in \Delta \right\}.$$

By Lemma 1.29, the transducer T_Δ obtained by restricting T_{Sh} to the domain R_Δ is both synchronous and sequential. For all $w = [w_1]\dots[w_n] \in ([\Lambda\widetilde{\Lambda}\Lambda^*])^{\geq 2}$, if $w' = T^N_\Delta(w)$ then $w' = [w'_1]\dots[w'_n]$ with $\pi(w_i) = \pi(w'_i)$ and $\rho(w'_i) = N + 2$ for all $i \in [1, n]$. Let r_i be the word containing the $N + 1$ first letters of w'_i, a straightforward induction on N proves that the picture p formed of the rows $r_1, \dots, r_n$ only has tiles in Δ. In particular, $T^N_\Delta(w)$ belongs to $([\Lambda^*\widetilde{\Lambda}])^* \cap R_\Delta$ if and only if $\pi(w)$ represent a picture p of width $N + 2$ such that $T(p) \subseteq \Delta$.

We now define more precisely the sequential rational graph $G = (T_a)_{a\in\Sigma}$ accepting L. For all $a \in \Sigma$, the transducer T_a is obtained by restricting the domain of T_Δ to the set of words representing pictures whose marked symbol on the second row is a, i.e. to the set $[(\Lambda \cup \widetilde{\Lambda})^*][(\Lambda^*\widetilde{a}\Lambda^*][(\Lambda \cup \widetilde{\Lambda})^*]^*$. By Lemma 1.29, T_a can be chosen synchronous and sequential. The set of initial vertices I is $[\#\widetilde{\#}\#^*]([\#\widetilde{\Gamma}\Gamma^*\#])^*[\#\widetilde{\#}\#^*]$ and the set of final vertices F is $[\#^*\widetilde{\#}]([\#\Gamma^*\widetilde{\#}])^*[\#^*\widetilde{\#}]$.

Let us prove that $L = L(G, I, F)$. Suppose that for some word $i \in I$, $T_w(i)$ belongs to F for $w = a_1 \dots a_n$. Then this implies that $T^n_\Delta(i)$ belongs to F and therefore, i represents a picture p of width $n+2$ such that $T(p) \subseteq \Delta$. As i belongs to I there exists a picture $q \in \Gamma^{**}$ such that $p = q_\#$. Moreover, the frontier of q is w and hence w belongs to L. We proved that $L(G, I, F) \subseteq L$. Conversely, let $w = a_1 \dots a_n$ be a word in L, there exist a picture q such that $w = \text{fr}(q)$ and $T(q_\#) \subseteq \Delta$. Let i be the word representing $q_\#$, it is easy to prove that $T_w(i)$ belongs to F. This shows that $L \subseteq L(G, I, F)$. □

Example 4.21. Figure 4.9 shows a part of the result of the previous construction when applied to the language $\{a^nb^n \mid n \geq 1\}$ as recognized by the tiling system of Figure 4.2. For convenience each vertex is represented by the corresponding

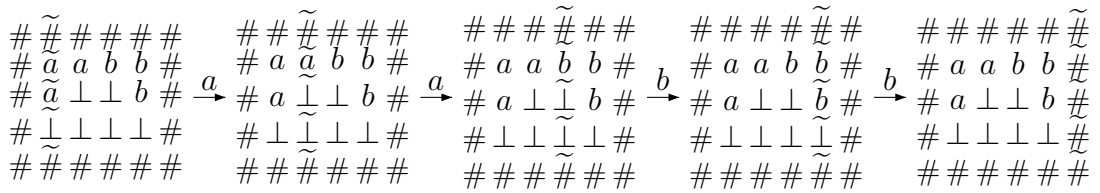


Figure 4.9: Connected component of a sequential synchronous graph accepting $\{a^n b^n \mid n \geq 1\}$.

picture, instead of the word coding for it. Also, only one connected component of the graph is shown. The other connected components all have the same linear structure: the degree of the graph is bounded by 1. The leftmost vertex belongs to the set I, and the rightmost to the set F, hence the word a^2b^2 is accepted.

Remark 4.22. In the case of synchronized transducers, it has been shown in Lemma 4.14 that I could be taken over a one letter alphabet without loss of generality. This does not seems to hold for sequential transducers as the proof we present relies on the expressiveness of the initial set of vertices. In fact, as shown in Proposition 4.40, the languages recognized by sequential synchronous graph from i^* are deterministic context-sensitive languages.

4.2.3 Rational graphs seen as automata

The structure of the graphs obtained in the previous section (propositions 4.11 and 4.20) is very poor. Synchronous graphs are by definition composed of a possibly infinite set of *finite* connected components. In the case of Proposition 4.20, we obtain an even more restricted family of graphs since both their in-degree and out-degree is bounded by 1. However, when considering accepted languages from a possibly infinite rational set of vertices, even this extremely restricted family accepts the same languages as the most general rational graphs, namely all context-sensitive languages. This is why, in order to compare the expressiveness of the different sub-families of rational graphs and to obtain graphs with richer structures, we need to impose structural restrictions.

In a first time, we consider graphs with a single initial vertex, but this restriction alone is not enough. In fact, both synchronized and rational graphs with a rational set of initial vertices accept the same languages as their counterparts with a single initial vertex.

Lemma 4.23. *For all rational graph (resp. synchronized graph) G and for any pair of rational sets I and F, there exists a rational graph (resp. a synchronized graph) G', a vertex i and a rational set F' such that $L(G, I, F) = L(G', \{i\}, F')$.*

Proof. Let $G = (T_a)_{a\in\Sigma}$ be a rational graph with vertices in Γ^* and let i be a symbol which does not belong to Γ and $\Gamma' = \Gamma \cup \{i\}$. For all $a \in \Sigma$, let T'_a be a transducer recognizing the rational relation $T_a \cup \{(i, w) \mid w \in T_a(I)\}$. Remark that if T_a is synchronized then T'_a can also be chosen synchronized. If $\varepsilon \notin L(G, I, F)$ then $F' = F$ else $F' = F \cup \{i\}$. It is straightforward to show that $L(G, I, F) = L(G', \{i\}, F')$. □

It follows from Proposition 4.11 and Lemma 4.23 that the synchronized rational graphs with one initial vertex accept the context-sensitive languages [Ris02].

Remark 4.24. It is fairly obvious that this result does not hold for synchronous graphs: indeed, the restriction of a synchronous rational graph to the vertices reachable from a single vertex is finite. Hence, the languages of synchronous graphs from a single vertex are rational. Similarly, as any rational language is accepted by a deterministic finite graph, it can also be accepted by a sequential synchronous graph with a single initial vertex.

Note that the construction of Lemma 4.23 relies on infinite out-degree to transform a synchronous graph with a rational set of initial vertices into a rational one with a single initial vertex. In order to obtain more satisfactory notions of infinite automata and their languages, we now restrict our attention to graphs of finite out-degree with a single initial vertex.

4.2.3.1 Rational graphs: finite out-degree and one initial vertex

We present a syntactical transformation of a synchronous rational graph with a rational set of initial vertices into a rational graph of finite out-degree with a unique initial vertex accepting the same language.

The construction relies on the fact that for a synchronous graphs to recognize a word of length $n > 0$, it is only necessary to consider vertices whose length is smaller than c^n (where c is a constant depending only on the graph). We first establish a similar result for tiling systems and conclude using the close correspondence between synchronous graphs and tiling systems established in Proposition 4.15.

Lemma 4.25. *For any tiling system $S = (\Gamma, \Sigma, \#, \Delta)$, if $p \in P(S)$ then there exists a (n, m)-picture p' such that $\mathrm{fr}(p) = \mathrm{fr}(p')$ and $n \leq |\Gamma|^m$.*

Proof. Let p' be a (n, m)-picture with $n > |\Gamma|^m$, and suppose that p' is the smallest picture in $P(S)$ with frontier $\mathrm{fr}(p)$. Let $l_1, \ldots, l_n$ be the rows of p'. As $n > |\Gamma|^m$ then there exists $j > i \geq 1$ such that $l_i = l_j$. Let p'' be the picture with rows $l_1, \ldots, l_i, l_{j+1}, \ldots, l_n$. It is easy to check that $T(p''_{\#}) \subset T(p'_{\#})$, we have that $p'' \in P(S)$ and as p'' has a smaller height than p' but the same frontier, we obtain a contradiction. □

We know from Remark 4.16 that for all synchronous rational graph $G = (T_a)_{a\in\Sigma}$ and two rational sets I and F, there exists a tiling system S such that $i \xrightarrow[G]{w} f$ with $i \in I$ and $f \in F$ if and only if there exists $p \in K$ such that $\mathrm{fr}\,(p) = w$ and p has height $|i| = |f|$. Hence, as a direct consequence of Lemma 4.25, one gets:

Lemma 4.26. *For all synchronous rational graph G and rational sets I and F, there exists $k \geq 1$ such that:*

$$\forall w \in L(G, I, F), \exists i \in I, f \in F \text{ such that } i \xrightarrow[G]{w} f \text{ and } |i| = |f| \leq k^{|w|}.$$

We can now present the construction of a rational graph of finite out-degree accepting from a single vertex the same language as a synchronous graph with a rational set of initial vertices.

Proposition 4.27. *For all synchronous rational graph G and rational sets I and F such that $I \cap F = \emptyset$, there is a rational graph H of finite out-degree and a vertex i such that $L(G, I, F) = L(H, \{i\}, F)$.*

Proof. According to Lemma 4.14, there exists a synchronous rational graph R described by a set of transducers $(T_a)_{a\in\Sigma}$ over Γ^* such that $L(G, I, F) = L(R, \#^*, F)$. Note that for all $w \in \#^*$ and $w' \in \Gamma^*$, if $w \xrightarrow[R]{} w'$ then w' does not contain $\#$. We now define a graph H such that $L(G, I, F) = L(H, \{i\}, F)$ for some vertex i of H. Let k be the constant involved in Lemma 4.26, T and T' two transducers realizing the rational relations $\left\{(\#^n, \#^{kn}) \mid n \in \mathbb{N}\right\}$ and $\{\#^n, \#^m \mid m \in [1, n]\}$ respectively. For all $a, b, c \in \Sigma$ and $u \in \Sigma^*$, H has the following sets of edges:

$$\begin{array}{rrcll}
\forall n \in \mathbb{N}, & u|\#^n & \xrightarrow{a} & ua|\, T \circ T(\#^n) & \text{(Type 1)} \\
\forall n \in \mathbb{N}, & bu|\#^n & \xrightarrow{a} & ua|\, T \circ T' \circ T_b(\#^n) & \text{(Type 2)} \\
\forall n \in \mathbb{N}, & bcu|\#^n & \xrightarrow{a} & ua|\, T' \circ T_b \circ T_c(\#^n) & \text{(Type 3)} \\
\forall w \in (\Gamma \setminus \{\#\})^*, & bcu|w & \xrightarrow{a} & ua|\, T_b \circ T_c(w) & \text{(Type 4)} \\
\forall w \in (\Gamma \setminus \{\#\})^*, & b|w & \xrightarrow{a} & T_b \circ T_a(w) & \text{(Type 5)} \\
 & |\# & \xrightarrow{a} & T \circ T' \circ T_a(\#) & \text{(Type 6)}
\end{array}$$

The graph H is clearly rational and of finite out-degree. We take $i = |\#$ as initial vertex and we claim that $L(R, \#^*, F) = L(H, \{i\}, F)$.

Remark that in H an edge of type 2 or 3 cannot be followed by edges of type 1, 2 or 3, and at most one edge of type 2 or 3 and of type 5 or 6 can be applied. Moreover, an edge of type 1 increases the length of the left part of the word by one, and an edge of type 4 decreases it by one. Also, in any accepting path, the

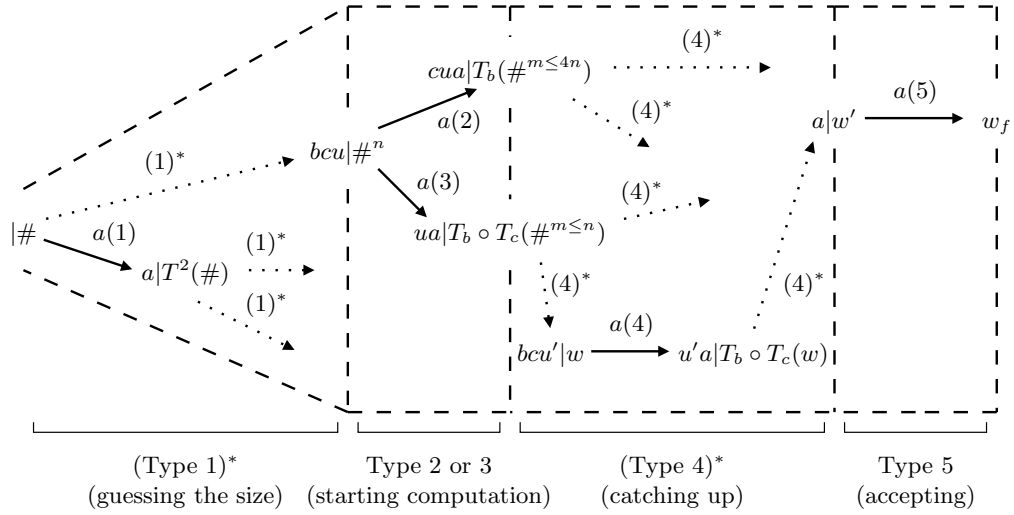


Figure 4.10: Schema of the construction in Proposition 4.27.

---

last edge is of type 5 or 6. Figure 4.10 illustrates the structure of the obtained graph.

Let us first prove that for every accepting path $i = c_0 \xrightarrow{a_1} \dots \xrightarrow{a_n} c_n \in F$ for $w = a_1 \dots a_n$ in H, there exists an accepting path $c'_0 \xrightarrow{a_1} \dots \xrightarrow{a_n} c'_n = c_n \in F$ for w in R. We distinguish three cases:

1. Case $n = 1$: the single edge must be of type 6. Hence $T_{a_1}(T'(T(\#))) \cap F$ is not empty. We can choose $c'_0 \in T'(T(\#))$ and $c'_1 \in T_{a_1}(c'_0)$, thus $w = a_1$ belongs to $L(R, \#^*, F)$.
2. Case $n = 2m$ with $m > 0$: the path in H is composed of m edges of type 1 followed by an edge of type 3, $m - 2$ edges of type 4 and finally an edge of type 5. Therefore, we have:

   (a) for all $i \in [1, m]$, $c_i \in a_1 \dots a_i | T^{2i}(\#)$,
   (b) for all $i \in [1, m-1]$ we have $c_{i+m} \in a_{2i+1} \dots a_{m+i} | c'_{2i}$ for some $c'_0 \in T^{2m} \circ T'(\#)$ and $c'_{2i} \in T_{a_1} \circ \dots \circ T_{a_{2i}}(c'_0)$),
   (c) $c_n = c'_n \in T_{a_1} \circ \dots \circ T_{a_n}(c'_0)$.

   We already fixed c'_k for all even index k between 0 and n. For all $i \in [1, m]$, we can choose c'_{2i-1} to be any word in $T_{a_{2i-1}}(c'_{2i-2})$ such that $c'_{2i} \in T_{a_{2i}}(c'_{2i-1})$ (which necessarily exists by construction of c'_{2i-2} and c'_{2i}). As c'_0 belongs to $\#^*$ and $c'_n = c_n \in F$, this means that $w \in L(R, \#^*, F)$.
3. Case $n = 2m + 1$ with $m > 0$: the path is composed of m edges of type 1 followed by an edge of type 2, $m - 1$ edges of type 4 and finally an edge of type 5. Therefore, we have:

(a) for all $i \in [1, m]$, $c_i \in a_1 \dots a_i | T^{2i}(\#)$,
(b) let $c'_0 \in T^{2m+1} \circ T'(\#)$, we have $c_{m+1} \in a_2 \dots a_{m+1} | c'_1$ for some $c'_1 \in T_{a_1}(c'_0)$,
(c) for all $i \in [1, m-1]$, we have $c_{i+m+1} \in a_{2i+2} \dots a_{m+i} | c'_{2i+1}$ for some $c'_{2i+1} \in T_{a_1} \circ \dots \circ T_{a_{2i+1}}(c'_0)$,
(d) $c_n = c'_n \in T_{a_1} \circ \dots \circ T_{a_n}(c'_0)$.

We already fixed c'_0, and c'_k for all odd index k between 0 and n. For all $i \in [1, m]$, we can choose c'_{2i} to be any word in $T_{a_{2i}}(c'_{2i-1})$ such that $c'_{2i+1} = T_{a_{2i+1}}(c'_{2i})$ (which necessarily exists by construction of c'_{2i-1} and c'_{2i+1}). As c'_0 belongs to $\#^*$ and $c'_n = c_n \in F$, this means that $w \in L(R, \#^*, F)$.

We proved that $L(H, \{i\}, F) \subseteq L(R, \#^*, F)$.

Let us now prove the converse, i.e. that for every accepting path $c_0 \in \#^* \xrightarrow{a_1} \dots \xrightarrow{a_n} c_n \in F$ in R with $w = a_1 \dots a_n$, there exists an accepting path for w in H. By Lemma 4.26 and by definition of k, we can assume that $|c_0| = |c_n| \leq k^n$. It is straightforward to prove that for all word x in $T_{a_1} \circ \dots \circ T_{a_n}(T^n \circ T'(\#))$, $i \xrightarrow[H]{w} x$. It remains to prove that c_n belongs $T_{a_1} \circ \dots \circ T_{a_n}(T^n \circ T'(\#))$. By definition of T and T', $T^n \circ T'(\#)$ is equal to $\{\#^i \mid i \in [1, k^n]\}$. Hence as $c_0 \in T^n \circ T'(\#)$, it follows that $c_n \in T_{a_1} \circ \dots \circ T_{a_n}(T^n \circ T'(\#))$. Therefore, $i \Longrightarrow_R^w c_n$ and as $c_n \in F$, w belongs to $L(H, \{i\}, F)$. This proves that $L(R, \#^*, F) \subseteq L(H, \{i\}, F)$. □

From Proposition 4.10 and Proposition 4.27, we deduce that the rational graphs of finite out-degree with one initial vertex accept all context-sensitive languages. This result was proved in [MS01] using the Penttonen normal form of context-sensitive grammars [Pen74].

Theorem 4.28. *The path languages of rational graphs of finite out-degree from a unique initial vertex to a rational set of final vertices are the context-sensitive languages.*

4.2.3.2 Synchronized graphs: finite out-degree and one initial vertex

We now consider the languages of synchronized graphs of finite out-degree with one initial vertex. First, we characterize them as the languages recognized by tiling systems with square pictures (i.e. for which there exists $c \in \mathbb{N}$ such that for all word $w \in L(S)$, there exists a (n, m)-picture in $P(S)$ with $n \leq cm$ and with frontier w). A slight adaptation of the construction of Proposition 4.27 gives the first inclusion.

Proposition 4.29. *Let $S = (\Gamma, \Sigma, \#, \Delta)$ be a tiling system with square pictures. There exists a synchronized rational graph of finite degree accepting $L(S)$ from one initial vertex.*

Proof. Let $G = (T_a)_{a\in\Sigma}$ be the synchronized graph obtained from S in Proposition 4.11. In the construction from the proof of Proposition 4.27, if we replace the transducer T by a transducer S realizing the synchronized relation $\{(\#^n, \#^{n+c}) \mid n \in \mathbb{N}\}$, we obtain a synchronized graph H, an initial vertex i and a set of final vertices F such that $L(H, i, F) = L(S)$. □

Before proceeding with the converse, we state a result similar to Lemma 4.26 for synchronized graphs *of finite out-degree* that states that when recognizing a word w *from a unique initial vertex* i, the vertices involved have a length at most linear in the size of w.

Lemma 4.30. *For any synchronized rational graph G of finite out-degree with vertices in Γ^* and for all vertex i, there exists a constant k such that for all w in $L(G, \{i\}, F)$, there exists a path from i to some $f \in F$, labeled by w, and with vertices of size at most $k \cdot |w|$.*

Proof. It follows from the definition of synchronized transducers that for every synchronized transducer of finite out-degree there exists $c \in \mathbb{N}$ such that $(x, y) \in T$ implies that $|x| \leq |y| + c$ (see [Sak03] for a proof of this result). We take k to be the maximum over the set of transducers defining G of these constants. The result follows by a straightforward induction on the size of w. □

The converse inclusion is obtained by remarking that the composition of the construction of Proposition 4.10 and Proposition 4.15 gives a tiling system with square pictures when applied to a synchronized graph of finite out-degree.

Proposition 4.31. *Let $G = (T_a)_{a\in\Sigma}$ be a synchronized graph of finite out-degree. For all initial vertex i and set of final vertices F, there exists a tiling system S with square pictures such that $L(S) = L(G, \{i\}, F)$.*

Proof. Let G', I' and F' be the synchronous graph and the rational set of initial and final vertices obtained by applying the constructions of Proposition 4.10 to G, $\{i\}$ and F. It is easy to show that for all word $w \in L(G', I', F')$, there exists $i' \in I'$ and $f' \in F'$ such that $i' \xrightarrow{w} f'$ with $|i'| = |f'| \leq k|w|$ where k is the constant of Lemma 4.30 for G. We conclude by Proposition 4.15, that states the existence of a tiling system S such that $L(S) = L(G', I', F')$. By Remark 4.13, S is a tiling system with square pictures. □

Putting together Proposition 4.31 and Proposition 4.29 and with the use of the simulation result from Theorem 4.6, we obtain the following theorem.

Theorem 4.32. *The languages accepted by synchronized graphs of finite out-degree from a unique vertex to a rational set of vertices are the context-sensitive languages recognized by non-deterministic linearly bounded machines with a linear number of head reversals.*

We conjecture that this class is strictly contained in the context-sensitive languages. However, few separation results exist for complexity classes defined by time and space restrictions (see for example [vM04]). In particular, the diagonalization techniques (see [For00]) used to prove that the polynomial time hierarchy (with no space restriction) is strict do not apply for lack of a suitable notion of *universal* LBM.

4.2.3.3 Bounding the out-degree

It is natural to wonder if the rational graphs still accept the context-sensitive languages when considering bounded out-degree. This is a difficult question, to which we only provide here a partial answer concerning the synchronized graphs of bounded out-degree.

It follows from Lemma 4.30 that the vertices used to accept a word w in a synchronized rational graph have a length at most linear in the length of w and therefore, can be stored on the tape of a LBM. Moreover if the graph is deterministic, we can construct a deterministic LBM accepting its language.

Proposition 4.33. *The language accepted by a deterministic synchronized graph from a unique initial vertex is deterministic context-sensitive.*

Proof. Let $G = (T_a)_{a\in\Sigma}$ be a deterministic synchronized graph over Γ, i a vertex and F a rational set of vertices. We define a deterministic LBM M accepting $L(G, \{i\}, F)$. When accepting $w = a_1 \dots a_{|w|}$, M starts by writing i on its tape. It successively applies $T_{a_1}, \dots, T_{a_{n-1}}$ and T_{a_n} to i. If the image of the current tape content by one of these transducers is not defined, the machine rejects. Otherwise, it checks whether the last tape content represents a vertex which belongs to F.

We now detail how the machine M can apply one of the transducers T of G to a word x in a deterministic manner. As T has a finite image, we can assume without loss of generality that $T = (\Gamma, Q, i, F, \delta)$ is in real-time normal form: $\delta \subset Q \times \Gamma \times \Gamma^* \times Q$ (see for instance [Ber79] for a presentation of this result). The machine enumerates all paths in T of length less than $c|x|$ in the lexicographic order where c is the constant associated to G in Lemma 4.30. For each such path ρ, it checks if it is an accepting path for input x, and in that case replaces x by the output of ρ.

The space used by M when starting with a word w is bounded by $(2c+1)|w|$. Moreover if M accepts w, then there exists a path from i to a vertex F in G labeled by w. Conversely, if w belongs to $L(G, \{i\}, F)$ then by Lemma 4.30, there exists a path in G from i to F with vertices of length at most $c|w|$ and by construction M accepts w. Hence, M is a deterministic linearly bounded Turing machine accepting $L(G, \{i\}, F)$. □

Remark 4.34. The result of Proposition 4.33 extends to any deterministic rational graph satisfying the property expressed by Lemma 4.30.

The previous result can be extended to synchronized graphs of bounded out-degree thanks to a uniformization result by Weber. First observe that a rational graph is of out-degree bounded by some constant k if and only if it is defined by transducers which associate at most k distinct images to any input word. The relations realized by these transducers are called k-valued rational relations.

Proposition 4.35 ([Web96]). *For any k-valued rational relation R, there exist k functional rational relations $F_1, \ldots, F_k$ such that $R = \bigcup_{i \in [1,k]} F_i$.*

Note that even if R is a synchronized relation, the F_i's are not necessarily synchronized. However, they still satisfy the inequality $|y| \leq |x| + c$ for all $(x, y) \in F_i$.

To any synchronized graph G with an out-degree bounded by k defined by a set of transducers $(T_a)_{a \in \Sigma}$, we associate the deterministic rational graph H defined by $(F_{a_i})_{a \in \Sigma, i \in [1,k]}$ where for all $a \in \Sigma$, $(F_{a_i})_{i \in [1,k]}$ is the set of rational functions associated to T_a by Proposition 4.35. According to Proposition 4.33 and to Remark 4.34, $L(H, \{i\}, F)$ is a deterministic context-sensitive language. Let π be the alphabetical projection defined by $\pi(a_i) = a$ for all $a \in \Sigma$ and $i \in [1, k]$, it is straightforward to establish that $\pi\left(L(H, \{i\}, F)\right) = L(G, \{i\}, F)$. As deterministic context-sensitive languages are closed under alphabetical projections, $L(G, \{i\}, F)$ is a deterministic context-sensitive language.

Theorem 4.36. *The language accepted by a synchronized graph of bounded out-degree from a unique initial vertex is deterministic context-sensitive.*

The converse result is not clear, for reasons similar to those presented in the previous section for synchronized graphs of finite degree. A precise characterization of the family of languages accepted by synchronized rational graphs of bounded degree would be interesting.

4.2.4 Notions of determinism

In this last part of the section on rational graphs, we investigate families of graphs which accept the deterministic context-sensitive languages. First of all, we examine the family yielded by the previous constructions when applied to deterministic languages. Then, we propose a global property over sets of transducers characterizing a sub-family of rational graphs whose languages are precisely the deterministic context-sensitive languages.

4.2.4.1 Non-ambiguous context-sensitive languages

In the proof of Proposition 4.11, given any tiling system we describe a way to build a synchronous rational graph accepting the same language. A natural question is thus to consider the family of graphs obtained in the deterministic case. When applying this construction to a deterministic tiling system S, one obtains a synchronous rational graph G (which is non-deterministic in general) and two rational sets of vertices I and F such that $L(G, I, F) = L(S)$, with the particularity that for all word w in L, there is *exactly one* path labeled by w leading from some vertex in I to a vertex in F: G is *non-ambiguous* with respect to I and F.

However, the converse is not granted: given a graph G and two rational sets I and F such that G is non-ambiguous with respect to I and F, we cannot ensure that $L(G, I, F)$ is a deterministic context-sensitive language. Indeed, if we apply the construction from Proposition 4.15, we get a tiling system such that no pair of distinct images have the same upper frontier. Remark that such a tiling system accepting at most one accepted picture p with frontier w for each w in $L(S)$ is not necessarily deterministic. Rather, it is quite straightforward to see that they correspond to the non-ambiguous linearly bounded machines (i.e. non-deterministic machines with at most one accepting computation per input word). The class of languages accepted by such machines is called USPACE(n), and it is not known whether this class coincides with either the context-sensitive or deterministic context-sensitive languages.

Rather than the desired result on deterministic language, this approach thus only yields the following intermediary statement.

Theorem 4.37. *Let L be a language, the following properties are equivalent:*

1. *L is a non-ambiguous context-sensitive language.*
2. *There is a rational graph G with non-ambiguous transducers and two rational sets I and F such that $L = L(G, I, F)$ and G is non-ambiguous with respect to I and F.*

This result only holds if one considers non-ambiguous transducers, i.e. transducers in which there is at most one accepting path per pair of words. The reason being that the ambiguity in the transducers would induce an ambiguity in the machine. However, we know that synchronized transducers can be made non-ambiguous (Cf. Remark 1.28). We obtain the follow corollary.

Corollary 4.38. *The languages of non-ambiguous synchronized graphs with rational sets of initial and final vertices are the non-ambiguous context-sensitive languages.*

Note that the non-ambiguity of a rational or synchronized graphs with respect to rational sets of vertices is undecidable, hence these classes of graphs accepting non-ambiguous context-sensitive languages are not recursive. However, one can make the following observation: as any rational function can be realized by a non-ambiguous transducer [Kob69, Sak03], the languages of deterministic rational graphs are, according to Theorem 4.37, non-ambiguous context-sensitive languages.

Corollary 4.39. *The path languages of deterministic rational graphs from an initial vertex i to a rational set F of vertices are non-ambiguous context-sensitive languages.*

Proof. Let $G = (T_a)_{a\in\Sigma}$ be a deterministic rational graph. For all $a \in \Sigma$, the relation F_a recognized by T_a is a function and according to [Kob69, Sak03], there exists a non-ambiguous transducer T'_a recognizing F_a. From Theorem 4.37, we know that for all $i \in I$, $L(G, \{i\}, F)$ is a non-ambiguous context-sensitive language. □

4.2.4.2 Globally deterministic sets of transducers

We just saw an attempt at characterizing natural families of graphs whose languages are the deterministic context-sensitive languages, which was based on a restriction of previous constructions to the deterministic case, but failed to meet its objective because of a slight nuance between the notions of determinism and non-ambiguity for tiling systems.

First, we naturally consider the class of sequential synchronous automata with an initial set of the form i^*. It is easy to check that when applying the construction of Proposition 4.15 to one of these automata, we obtain a deterministic tiling system.

Proposition 4.40. *The languages of sequential synchronous graphs from i^* are deterministic context-sensitive languages.*

The converse result seems difficult to prove due to the local nature of the determinism involved in this class. Hence, we consider a *global* property of the set of transducers characterizing a rational graph, so as to ensure that each accepting path corresponds to the run of a deterministic linearly bounded machine on the corresponding input, or equivalently that each accepting path corresponds to a picture recognized by a deterministic tiling system and whose upper frontier is the path label under consideration.

For any rational language L, we write T_L the minimal synchronous transducer recognizing the identity relation over L.

Definition 4.41. Let T be a set of synchronous transducers over Γ. We say T is *globally deterministic* with respect to two rational languages I and $F \subseteq \Gamma^*$ if all transducers in T are deterministic[4] and for all pair of transducers $T_1 \in T \cup \{T_I\}$ and $T_2 \in T \cup \{T_F\}$, and all pair of control states $q_1 \in Q_{T_1}$ and $q_2 \in Q_{T_2}$, there is at most one b such that

$$q_1 \xrightarrow[T_1]{a/b} q_1' \wedge q_2 \xrightarrow[T_2]{b/c} q_2' \text{ for some } a, c \in \Gamma,\ q_1' \in Q_{T_1},\ q_2' \in Q_{T_2}.$$

Intuitively, this condition states that, whenever a part of the output of one transducer can be read as input by a second transducer, there is only one way to add a letter to this word such that it is still compatible with both transducers. This property of sets of transducers is trivially decidable, since it is sufficient to check the above condition for all pair of control states of transducers in $(T \cup \{T_I\}) \times (T \cup \{T_F\})$. This allows us to capture a sub-family of rational graphs whose languages are the deterministic context-sensitive languages.

Theorem 4.42. *Let L be a language, the following two properties are equivalent:*

1. *L is a deterministic context-sensitive language.*
2. *There is a synchronous rational graph G and two rational sets I and F such that $L = L(G, I, F)$ and G is globally deterministic between I and F.*

Proof. Let $G = (T_a)_{a\in\Sigma}$ be a synchronous rational graph which is globally deterministic between I and F. The graph $H = (T'_a)_{a\in\Sigma}$ obtained by applying Lemma 4.14 to G is such that $L(H, i^*, f^*) = L(G, I, F)$. Moreover, H is globally deterministic between i^* and f^*.

We will show that the construction of Proposition 4.15, when applied to a rational graph H between i^* and f^* yields a deterministic tiling system. Suppose that this is not the case. Then, by definition of a non-deterministic tiling system, there must be words u, v_1 and v_2 with $v_1 \neq v_2$ such that the two-rows pictures p_1 and p_2 with first row $\#u\#$ and second row $\#v_1\#$ and $\#v_2\#$ respectively only have tiles in Δ. Since $v_1 \neq v_2$, let i be the smallest index such that $v_1(i) \neq v_2(i)$. Let $v_1(i) = xp$, $v_2(i) = x'p'$.

According to the construction of Proposition 4.15, there are two transducers T_a and T_b such that

$$q_a \xrightarrow[T_a]{y/x} p \wedge q_b \xrightarrow[T_b]{x/z} q_b' \wedge q_a \xrightarrow[T_a]{y/x'} p' \wedge q_b \xrightarrow[T_b]{x'/z'} q_b''$$

for some symbols $y, y', z, z' \in \Gamma$ and control states q_a, q_b, q_b' and q_b''. As T_a is deterministic, if x is equal to x', then $p = p'$ and $v_1(i) = v_2(i)$. Hence $x \neq x'$, and the above relations contradicts the global determinacy of H.

[4] i.e. whenever $q \xrightarrow{a/b} q'$ and $q \xrightarrow{c/d} q''$ with $q' \neq q''$, it implies $(a, b) \neq (c, d)$.

Conversely, let L be any deterministic context-sensitive language. By Theorem 4.7, there exists a deterministic cellular automaton $C = (\Gamma, \Sigma, \perp, \delta, [,])$ recognizing L. We will build two rational languages I and F and a set of transducers T globally deterministic with respect to I and F such that $L(G, I, F) = L$ where G is the rational graph defined by T. The work alphabet of T is $\Gamma' = \Sigma \cup \{[,]\} \cup \delta$. The set of control states of transducer $T_a \in T$ is $\{q_0^a\} \cup \{q_{AB}^a \mid A, B \in \Gamma \cup \{[]\}\}$, where q_0^a is the unique initial state. Its transitions are:

$$\forall a, b \in \Sigma, \quad q_0^a \xrightarrow{[/a} q_{[a}^a \quad \text{and} \quad q_0^a \xrightarrow{b/a} q_{ba}^a$$
$$\forall d_1 = ([, A, B, A') \in \delta, \quad q_{[A}^a \xrightarrow{[/d_1} q_{[A'}^a$$
$$\forall d_1 = (A, B, C, B'), d_2 = (B, C, D, C') \in \delta, \quad q_{BC}^a \xrightarrow{d_1/d_2} q_{B'C'}^a$$

The terminal states of T_a are $q_{[\perp}$ and $q_{\perp\perp}$. Now let $I = ([])^*$ and $F = \Sigma R^*$ where $R = \{(a, b,], b') \in \delta \mid a, b, b' \in \Gamma\}$. By construction and since C is deterministic, T is globally deterministic with respect to I and F. Let us verify that $L(G, I, F) = L$.

$L(G, I, F) \subseteq L$: let $v_0, \ldots, v_n$ be an w-labeled accepting path in G, with $v_0 = ([])^{m+1} \in I$, $v_n \in F$, $n = |w|$ and $v_i(1) = w(i)$ for all $i > 0$. We call p the $m \times n$ picture whose columns are $v_0, \ldots, v_n, v_{n+1}$ (with $v_{n+1} = (])^{m+1}$), and $u_0 \ldots u_m$ the rows of p. Denote by π the morphism on $(\Sigma \cup \{[,]\} \cup \delta)^*$ defined as $\pi(a) = a$ for all $a \in \Sigma$, $\pi([) = [$, $\pi(]) =]$ and $\pi(d) = B$ for all $d = (A, B, C, D) \in \delta$. By construction of T, for all $i \in [2, m]$ and $j \in [1, n]$, we have $\pi(u_i(j)) = B'$ if and only if there exist letters A, B and $C \in (\Gamma \cup \{[,]\})$ such that $u_{i-1}(j) = (A, B, C, B') \in \delta$, or in other terms:

$$\pi(u_{i-1}(j-1)) = A,\ \pi(u_{i-1}(j)) = B,\ \pi(u_{i-1}(j+1)) = C\ \wedge \begin{array}{|c|c|c|}\hline A & B & C \\ \hline & B' & \\ \hline \end{array} \in \delta.$$

This implies that for all $i \in [2, m]$, there is a transition in C between the configurations represented by u_{i-1} and u_i according to δ, in other words $(u_1, \ldots, u_m)$ is a run of C between u_1 and u_m. As furthermore $\pi(u_1) = [w]$ and by definition of the final states of T, there must exists a transition in C between $\pi(u_m)$ and $[\perp^n]$, w is accepted by C.

$L \subseteq L(G, I, F)$: let $w \in L$, and $(u_1 = [w], u_2, \ldots, u_m = [\perp^n])$, with $n = |w|$, be an accepting run of C for w. Define u'_i, $i \in [1, m-1]$, as the word on δ^* defined as follows:

$$u'_i(1) = [\,, \quad u'_i(n+1) =] \ \text{ and } \ \forall j \in [2, n+1],\ u'_i(j) = d_{i,j} \text{ with } \big(u_i(j-1), u_i(j), u_i(j+1), u_{i+1}(j)\big) \in \delta.$$

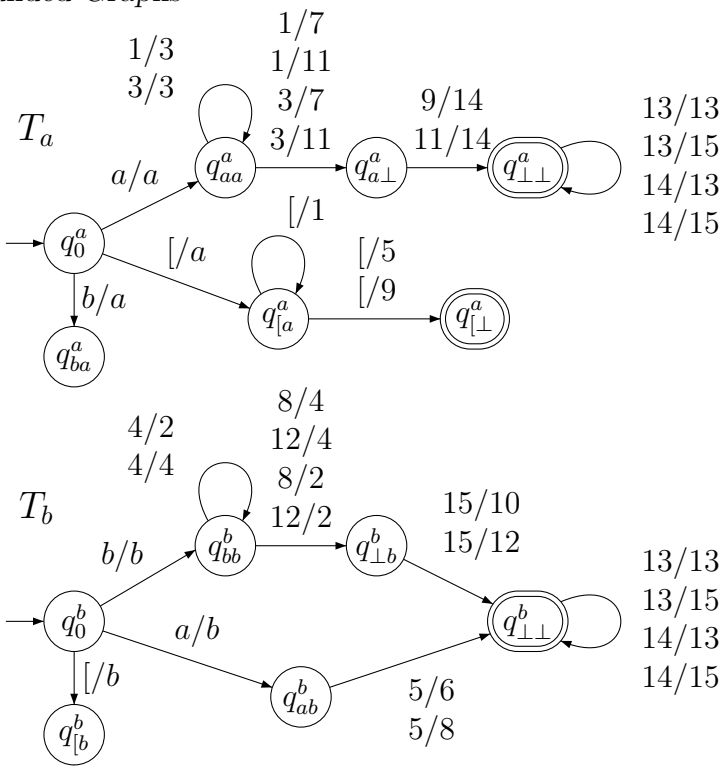


Figure 4.11: Transducers of a globally deterministic synchronous graph accepting $\{a^n b^n \mid n \geq 1\}$.

In other words, $u'_i = [d_1 \dots d_n]$, where each d_j is the unique rule of δ used at index j to transform u_i in u_{i+1}. Now consider the picture p formed of all rows u'_1 to u'_{m-1}, and let $v_1, \dots, v_{n+2}$ be the columns of p. It is quite straightforward to prove from the construction of T that $([v_1, w(1)v_2) \in T_{w(1)}$ and $(w(j)v_{j+1}, w(j+1)v_{j+2}) \in T_{w(j+1)}$ for all $j \in [1, n-1]$. Also $[v_1 \in I$ and $w(n)v_{n+1} \in F$, and hence w labels an accepting path in G. □

Example 4.43. Figure 4.11 illustrates the previous construction when applied to the deterministic cellular automaton of Example 4.5 recognizing the language $\{a^n b^n \mid n \geq 1\}$. The transitions of the transducers are labeled by either letters in $\Sigma \cup \{[\}$ or cellular automata transitions denoted by their numbers (see Figure 4.4).

4.3 Linearly Bounded Graphs

Among all the methods we presented in Section 2.1 to give finite presentations of infinite families of graphs, and additionally to other representations, both prefix-

recognizable graphs and Turing graphs have alternative definitions as transition graphs and Cayley-type graphs of rewriting systems. Rational graphs, however, lack a characterization as transition graphs, which makes the correspondence with their languages less clear, as we just saw. In this section, we are thus interested in defining a suitable notion of transition graphs of linearly bounded Turing machines which would naturally accept the context-sensitive languages, and in determining some of their structural properties.

We consider the transition graphs of linearly bounded machines according to the definition of transition graphs of arbitrary machines given in Section 2.1.1.1. For convenience, we call such graphs *linearly bounded graphs*. A similar family was studied in [KP99, Pay00], where the configuration graphs of LBMs up to weak bisimulation is studied. However, it provides no formal definition associating LBMs to a family of *real-time* graphs (without edges labeled by silent transitions) representing their observable computations.

To further illustrate the suitability of our notion, we provide two alternative definitions of linearly bounded graphs. First, we prove that they are isomorphic to the Cayley-type graphs of length-decreasing rewriting systems. The second alternative definition directly represents the edge relations of a linearly bounded graph as a certain kind of context-sensitive transductions. This allows us to straightforwardly deduce structural properties of linearly bounded graphs, like their closure under synchronized product (which was already known from [KP99]) and under restriction to a context-sensitive set of vertices.

4.3.1 Definition

4.3.1.1 LBM Transition Graphs

We define linearly bounded graphs as the closure under isomorphism of the family of transition graphs of *normalized* LBMs, as defined in Section 2.1.1.1.

Remark 4.44. For simplicity, we will only consider linearly bounded machines which insert a new tape cell each time a letter is read. More relaxed forms where a cell deletion or rewriting can occur during an input may be considered without any consequence for the results. Similarly, rules which do not move the read head can be allowed.

We first show that, in the case of linearly bounded machines, imposing normalization does not decrease the expressiveness of the model. We recall that a linearly bounded machine is normalized if, from any configuration, either only ε-transitions or no ε-transitions are possible. Note that the set of external configurations of a normalized LBM is a rational set, since the fact that a configuration is external can be seen simply by looking at the set of currently activated transition rules.

Proposition 4.45. *Every labeled linearly bounded machine can be normalized without changing the accepted language.*

Proof. Let M be a LBM, we build a normalized LBM M' equivalent to M. Without loss of generality, we will assume M has no ε-labeled cycle containing an accepting configuration. This property is very similar to 'real-timeness' (Cf. Proposition 1.10). We also suppose that the initial state q_0 of M is an external control state.

First, M' has at least all control states and ε-transitions of M, and the same initial state q_0. For all transition rule $pB \xrightarrow{a} qAB$ of M with $p \neq q_0$, we add a new control state p' and a rule $p'B \xrightarrow{a} qAB$, as well as ε-rules allowing to go from p to p' without moving the read head. We then take as set of external states for M' the set of all such new states p', which by definition only allow Σ-transitions, as well as q_0 and the states of M which are source of no transition. This way, the condition on control states is met. It remains to ensure that all final states of M' are external. To achieve this, whenever a final state is encountered, we transmit this information throughout each possible ε-transition by marking the control state. The marking is reset whenever a labeled transition is performed. By the previous assumption on the absence of ε-cycles containing accepting configurations, all computations reaching a final configuration will eventually reach an external state of M'. It thus only remains to declare all marked external states as accepting for M' to be normalized and equivalent to M. □

This result may be easily extended to the whole class of Turing machines. From this point on, unless otherwise stated, we will only consider normalized LBMs.

Example 4.46. Figure 4.12 shows the transition graph of the normalized LBM M from Example 1.9. This machine accepts the language $\{(a^n b^n)^+ \mid n \geq 1\}$, which is also the language of paths of the graph between vertex $[q_0]$ and the set $[b^* q_2 b]$. For the sake of clarity, only the part of the graph reachable from configuration $[q_0]$ is shown. We will see in Section 4.3.2 that this subgraph is still a linearly bounded graph.

4.3.1.2 Alternative definitions

This section provides two alternative definitions of linearly bounded graphs. In [CK02a], it is shown that all previously mentioned families of graphs can be expressed in a uniform way in terms of Cayley-type graphs of certain families of rewriting systems. We show that it is also the case for linearly bounded graphs, which are the Cayley-type graphs of length-decreasing rewriting systems. The

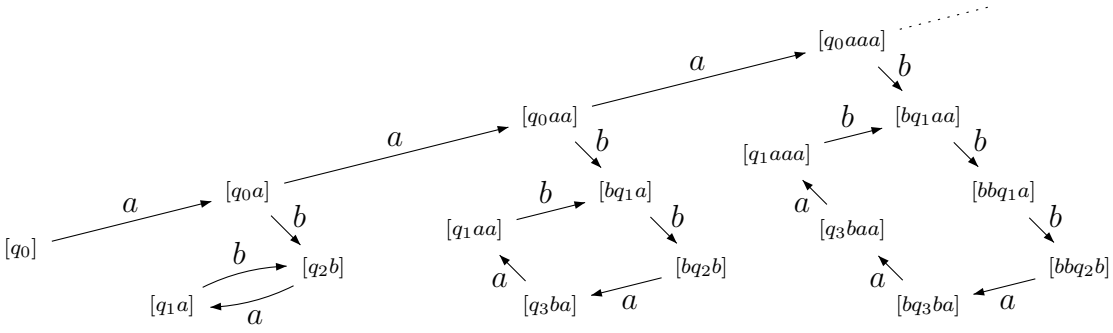


Figure 4.12: The transition graph of a labeled LBM accepting $\{(a^n b^n)^+ \mid n \geq 1\}$.

---

second alternative definition we present changes the perspective and directly defines the edges of linearly bounded graphs using incremental context-sensitive transductions. This variety of definitions will allow us to prove in a simpler way some of the properties of linearly bounded graphs.

Cayley-type graphs of decreasing rewriting systems. Cayley-type graphs of rewriting systems are defined in Section 2.1.1.2. The family of rewriting systems we consider is the family of *finite length-decreasing word rewriting systems*, i.e. rewriting systems with a finite set of rules of the form $l \to r$ with $|l| \leq |r|$, which can only preserve or decrease the length of the word to which they are applied. The reason for this choice is that the derivation relations of such systems coincide with arbitrary compositions of labeled LBM ε-rules.

Example 4.47. Figure 4.13 shows the Cayley-type graph of a simple decreasing rewriting system.

Incremental context-sensitive transduction graphs. We finally show that it is also possible to give a definition of linearly bounded graphs as the computation graphs of a certain family of LBMs, or equivalently as the graphs defined by a certain family of context-sensitive transductions (see Section 2.1.1.1 for the definition of computation graphs).

A binary relation R is recognized by a LBM M if the language $\{u\#v \mid (u, v) \in R\}$ where $\#$ is a fresh symbol is accepted by M. However, this type of transductions generates more than linearly bounded graphs. Even if we only consider linear relations (i.e relations R such that there exists c and $k \in \mathbb{N}$ such that $(u, v) \in R$ implies $|v| \leq c \cdot |u| + k$), we obtain graphs accepting the languages recognizable in exponential space (NSPACE[2^n]) which strictly contain the context-sensitive languages (cf Theorem 1.12). We need to consider relations

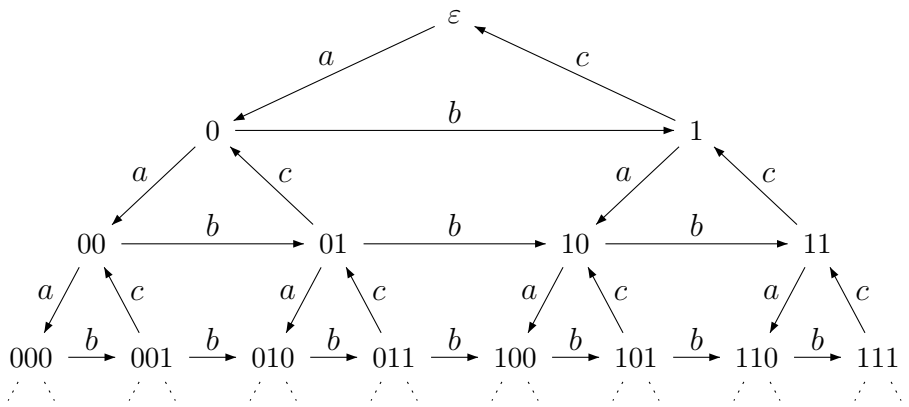


Figure 4.13: Cayley-type graph of the rewriting system $R = \{a \to 0, b \to b, 0b \to 1, 1b \to b0, c \to c, 1c \to \varepsilon\}$.

---

for which the length difference between a word and its image is bounded by a certain constant. Such relations can be associated to LBMs as follows:

Definition 4.48. A *k-incremental context-sensitive transduction* T over Γ is defined by a LBM recognizing a language $L \subseteq \{u\#v \mid u, v \in \Gamma^* \text{ and } |v| \leq |u| + k\}$ where $\#$ does not belong to Γ. Relation T is defined as $\{(u, v) \mid u\#v \in L\}$.

The synchronized relations of finite image (i.e for u there are finitely many v such that $(u, v) \in R$) provide a first example of k-incremental context-sensitive transductions.

Proposition 4.49. *For any synchronized relation R of finite image, there exists a constant $k \in \mathbb{N}$ such that R is a k-incremental context-sensitive transduction.*

Proof. Let $R \subseteq \Gamma^* \times \Gamma^*$ be a synchronized relation of finite image. It follows from the definition of synchronized relations that R is equal to a finite union:

$$\left(\bigcup_{i \in I} S_i \cdot (A_i \times \varepsilon)\right) \cup \left(\bigcup_{j \in J} S_j \cdot (\varepsilon \times B_j)\right)$$

where for all $k \in I \cup J$, S_k is a synchronous relation and for all $i \in I$ and $j \in J$, A_i and B_j are rational subsets of Γ^*.
As R has a finite image, for all $j \in J$ the set B_j is necessarily finite. Let k be the maximal length of a word in $\bigcup_{j \in J} B_j$, it is easy to check that for all pair of words $(u, v) \in R$, $|v| \leq |u| + k$. Moreover the language $\{u\#v \mid (u, v) \in T\}$ is context-sensitive. Hence T is a k-incremental context-sensitive transduction. □

The following proposition states that incremental context-sensitive transductions of a given level form a boolean algebra.

Proposition 4.50. *For all k-incremental context-sensitive transductions T and T' over Γ^*, $T \cup T'$, $T \cap T'$ and $\overline{T} = E_k - T$ (where E_k is $\{(u,v) \mid 1 \leq |v| \leq |u|+k\}$) are incremental context-sensitive transductions.*

Proof. The closure under union follows from that of context-sensitive languages. The proof of closure under complementation is a straightforward consequence of the closure under complementation of context-sensitive languages (cf Theorem 1.11). Let $T \subset \Gamma^* \times \Gamma^*$ be a k-incremental context-sensitive transduction. By definition, the set $L = \{u\#v \mid (u,v) \in T\}$ is context-sensitive. It is straightforward to check that the set $L' = \{u\#v \mid (u,v) \in E_k - T\}$ is equal to:

$$\overline{L} \cap \{u\#v \mid |v| \leq |u| + k\}.$$

As the context-sensitive languages are closed under complement and intersection, L' is context-sensitive. Hence, $\bar{T}$ is a k-incremental context-sensitive transduction. □

We consider, as usual, the closure under isomorphism of the family of incremental context-sensitive transduction graphs.

Example 4.51. The linearly bounded graph of Figure 4.12 can be seen as the transduction graph of the following set of incremental context-sensitive transductions:

$$T_a = \{(\#a^n, \#a^{n+1}) \mid n \geq 0\} \cup \{(b^m a^n, b^{m-1} a^{n+1}) \mid m \geq 1, n \geq 0\},$$
$$T_b = \{(\#a^n, a^{n-1} b) \mid n \geq 1\} \cup \{(a^m b^n, a^{m-1} b^{n+1}) \mid m \geq 1, n \geq 0\}.$$

As for the Cayley-type graph of Ex. 4.47, it can be seen as the transduction graph of the set of incremental context-sensitive transductions $\{T_a, T_b, T_c\}$, where T_a adds a 0 and T_c removes a 1 to the right of a binary number, and T_b implements binary increment.

Length-preserving context-sensitive transductions have already been extensively studied in [LST98]. In the rest of this presentation, unless otherwise stated, we will only consider 1-incremental transductions without loss of generality regarding the obtained family of graphs.

Equivalence of all definitions We now prove that both families of Cayley-type graphs of decreasing rewriting systems, and incremental context-sensitive transduction graphs define precisely the family of linearly bounded graphs, up to isomorphism (i.e. up to vertex renaming).

Theorem 4.52. *For any graph G, the following statements are equivalent:*

1. *G is isomorphic to the transitions graph of a labeled LBM,*
2. *G is isomorphic to the Cayley-type graph of a finite length-decreasing system,*
3. *G is isomorphic to a context-sensitive transduction graph.*

Proof. $1 \implies 2$: Let $M = (Q, \Sigma, \Gamma, \delta, q_0, F, [,])$ be a normalized labeled linearly bounded Turing machine, with $\Sigma \cap \Gamma = \emptyset$. As M is normalized, its configurations can be partitioned into Q_Σ and Q_ε (see Section 2.1.1.1). Let $\Gamma' = \Gamma \cup \{[,]\}$. We build a finite length-decreasing rewriting system R whose Cayley-type graph is the transition graph of M. Let the alphabet of R be $\Delta = \Sigma \cup \Gamma' \cup (\Gamma' \times Q) \cup S$, where $S = \{v_a, v'_a, s_a \mid a \in \Sigma\}$ is a new set of symbols disjoint from Γ and Q. Elements (x, q) of $\Gamma' \times Q$ will be noted x_q. For convenience, for any set X, $X_\bullet$ will denote $X \cup (X \times Q)$, and $x_\bullet$ any symbol in $\{x\}_\bullet$.

There are several important points which the rules of R must ensure:

1. Only words of the form $(\varepsilon \cup [_\bullet)\Gamma^*_\bullet(\varepsilon \cup]_\bullet)$ should be stable configurations:

$$x[_\bullet \to [_\bullet, \qquad]_\bullet y \to]_\bullet, \qquad s \to s$$

   for all $x \in \Delta,\ y \in \Delta \setminus \Sigma,\ s \in \Sigma \cup S \cup (\Gamma' \times Q_\varepsilon)$.

2. When a letter in Σ is added to the right of an irreducible word u, one should ensure that u actually represents a legal configuration, i.e. that u is of one of the forms $[_q\Gamma^*]$, $[\Gamma^* A_q \Gamma^*]$ or $[\Gamma^*]_q$ for $A \in \Gamma$, $q \in Q$:

$$]a \to v_a], \qquad]_q a \to {v_a}'\,]_q, \qquad A_q v_a \to {v_a}' A_q,$$
$$A{v_a}' \to {v_a}' A, \qquad [{v_a}' \to [s_a$$

   for all $a \in \Sigma, q \in Q, A \in \Gamma$.

3. Finally, once it has been made sure that the word represents a legitimate LBM configuration, one should simulate an insertion operation followed by any number of ε-transitions of the LBM:

$$s_a A \to A s_a, \qquad s_a B_p \to C_q B$$

   for all $a \in \Sigma,\ A, B, C \in \Gamma$ and $pB \xrightarrow{a} qCB \in \delta$, and

$$A_p C \to B C_q, \qquad C A_p \to C_q B \qquad A_p C \to C_q$$

   for all $pA \xrightarrow{\varepsilon} qB+,\ pA \xrightarrow{\varepsilon} qB-,\ pA \xrightarrow{\varepsilon} q \in \delta$ (respectively).

There is an edge $u \xrightarrow{a} v$ in the Cayley-type graph G_R of R if and only if u and v are words representing valid configurations of M from which no ε-transition can be performed, i.e. observable configurations, and there exists a sequence of transitions labeled by $a\varepsilon^*$ of M by which u reaches v. There is a bijection between the edges and vertices of G_R and the transition graph of M, hence these two graphs are isomorphic.

2 $\implies$ 3: Let R be a finite length-decreasing rewriting system, G_R its Cayley-type graph. For each letter a, we will show that the relation

$$T_a \;=\; \{(ua, v) \mid u \xrightarrow[G_R]{a} v\} = \{(ua, v) \mid u, v \in \mathrm{NF}(R) \wedge ua \xrightarrow[R]{*} v\}.$$

is an incremental context-sensitive transduction by building a LBM $M_a = (Q, \Sigma, \Gamma, \delta, q_0, F, [,])$ recognizing T_a.

For all pair (ua, v), M_a starts in configuration $ua\#v$, and first has to check that u is a normal form of R by verifying that it contains no left-hand side of any rule in R. Second, M_a simulates the derivation of R on u, applying one rewriting rule at a time until a normal form is reached. Due to non-determinism, there might be unsuccessful runs, but the pair is accepted if and only if one run reaches configuration $v\#v$, meaning that R can normalize ua into v. Hence, a pair (ua, v) is in T_a if and only if $(u, a, v) \in G_R$, meaning that the transduction graph of $(T_a)_{a\in\Sigma}$ is isomorphic to G_R.

3 $\implies$ 1: Let $T = (T_a)_{a\in\Sigma}$ be a finite set of incremental context-sensitive transductions defining a graph G_T, each T_a being recognized by a LBM M_a. We informally describe a normalized labeled LBM M whose transition graph G_M is isomorphic to G_T.

Let q be the unique external control state of M, M should have a run labeled by $a\varepsilon^*$ between configurations qu and qv whenever $(u, v) \in T_a$, or equivalently whenever the word $u\#v$ is accepted by M_a. This is done as follows. First, starting from configuration qu, M should perform an a-labeled transition, increasing its available tape space by 1, and step into an internal control state. It should then guess a word v, and write $u\#v$ on its tape. Since T_a is incremental, this can be done using no more than $|u| + 1$ tape cells by writing two symbols in each cell. Then, M simulates the LBM M_a on input word $u\#v$, while keeping an intact copy of v on the tape (this can again be done by a simple alphabet encoding). If the simulated run of M_a succeeds, M steps into external configuration qv by restoring the saved copy of v on the tape, otherwise it loops in a non-accepting internal state. By this construction, there is an edge (qu, a, qv) in G_M if and only if there is an edge (u, a, v) in G_T, hence both graphs are isomorphic. □

This shows that the three families of graphs presented in this section actually all define the same family of graphs, namely the linearly bounded graphs. This

variety of definitions will allow us to prove in a simpler way some of the properties of linearly bounded graphs.

4.3.2 Structural properties

Now that the family of linearly bounded graphs has been defined using three different formalisms, we can easily deduce some of their structural properties. In particular, we look at the languages accepted by linearly bounded graphs and some of their closure properties. We also give some insight about the relation between linearly bounded graphs and deterministic context-sensitive languages, and conclude with a few logical properties. But first, we compare our notions to related work.

4.3.3 Comparison with existing work

We now give a precise comparison of linearly bounded graphs with the restriction of Turing graphs ([Cau03]) to the linearly bounded case, and with the configuration graphs considered in [KP99].

Configuration graphs. In [KP99], the configuration graphs of a class of offline linearly bounded machines very similar to our labeled LBM are considered. However, they include in their definition a restriction to the set of configurations reachable from the initial configuration. Therefore, their class of configuration graph is incomparable to ours. As we will see in Proposition 4.56, the linearly bounded graphs are closed under restriction to the set of vertices reachable from a given vertex. Hence, it is not necessary to impose this restriction directly in the definition of the configuration graph.

Apart from this restriction, our family of configuration graphs coincides with the configuration of graph of [KP99] and to the configuration graphs from [Cau03] in the case of linearly bounded Turing machines.

Transition graphs. Knapik and Payet do not consider transition graphs: instead of characterizing the ε-closure of configuration graphs, they prove a closure property of this family up to weak bisimulation [Mil89], which is not a structural characterization. Nevertheless, it is straightforward to prove that their results can be extended to the family of transition graphs considered up to isomorphism, as mentioned in Section 4.3.2.

In [Cau03], Caucal defines the transition graphs of Turing machines from their configuration graphs using the very general notion of ε-closure: wherever there is a path labeled by $\varepsilon^* a \varepsilon^*$ in the configuration graph, there is an edge labeled by a in its ε-closure, for all letter a. Furthermore, the definition allows the restriction to

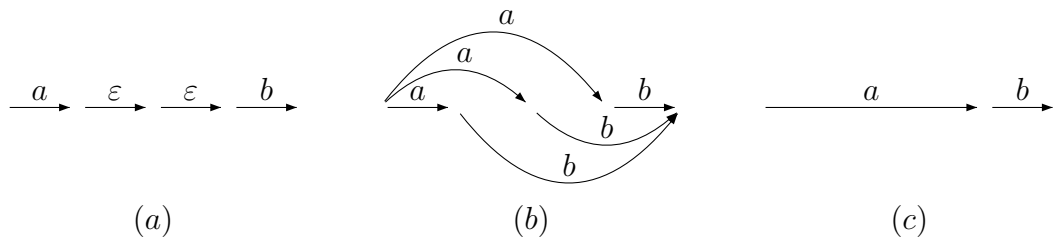


Figure 4.14: (a) Normalized LBM configuration graph. (b) Transition graph of (a) as defined in [Cau03] (with no restriction on vertices). (c) Linearly bounded graph associated to (a).

---

an arbitrary rational set of vertices (potentially the whole set of vertices). Figure 4.14 illustrates on a small configuration graph the difference between these two approaches.

Our approach has two main advantages. First, the ε-closure operation as defined by Caucal may give rise to spurious non-determinism in the obtained graphs. This additional complexity is artificial and is eliminated using our definition. Furthermore, our notion is purely structural, and does not rely on the naming of vertices. But interestingly, we can show that both families still coincide up to isomorphism: for all linearly bounded machine M and rational set R, one can define a machine M' such that the transition graph of M following Caucal's definition with respect to R is isomorphic to the linearly bounded graph corresponding to M' in our framework.

Proposition 4.53. *The family of linearly bounded graphs coincides, up to isomorphism, with the restriction of Turing graphs to the linearly bounded case.*

Proof. Let M be a normalized LBM, C its configuration graph, R its (rational) set of external configurations and G its transition graph defined as

$$G = \{c \xrightarrow{a} c' \mid c, c' \in R \wedge c \xrightarrow{a\varepsilon^*} c' \in C\}.$$

Consider the graph G' defined as the restriction to R of the ε-closure of C.

$$G' = \{c \xrightarrow{a} c' \mid c, c' \in R \wedge c \xrightarrow{\varepsilon^* a \varepsilon^*} c' \in C\}.$$

The graphs G and G' are equal, since by definition of external vertices, they have no outgoing ε-transitions.

Conversely, let M be a (not necessarily normalized) LBM, C its configuration graph, and R a rational set of configurations of M. The binary relation $\xrightarrow{\varepsilon^* a \varepsilon^*}$ in $C \times C$ is a 1-incremental context-sensitive transduction. Hence the graph G defined as the restriction to R (see Proposition 4.56) of the ε-closure of C is a linearly bounded graph. □

4.3.3.1 Languages

It is quite obvious that the language of the transition graph of a LBM M between the vertex representing its initial configuration and the set of vertices representing its final configurations is the language of M. In fact, the choice of initial and final vertices has no importance in terms of the family of languages one obtains.

Proposition 4.54. *The languages of linearly bounded graphs between an initial vertex i and a finite set F of final vertices are the context-sensitive languages.*

Proof. Let G be a linearly bounded graph labeled by Σ defined by a family $(T_a)_{a\in\Sigma}$ of 1-incremental context-sensitive transductions. We will prove that $L(G, i, F)$ is context-sensitive even if F is a context-sensitive set.

Let $\# \notin \Sigma$ be a new symbol, we consider the graph $\overline{G}$ obtained from G by adding a loop labeled by $\#$ on each vertex in F. Obviously, $\overline{G}$ is a linearly bounded graph as $\xrightarrow{\#} = \{(f, f) \mid f \in F\}$ is a 0-incremental context-sensitive transduction. By Theorem 4.52, G is the transition graph of a LBM M. Let c_0 be the configuration of M corresponding to the vertex i in G. Consider the LBM M' whose initial configuration is i (see Rem. 1.7) and whose set of final states are the states that appear in the left-hand side of a $\#$-rule (without loss of generality, we can assume that the $\#$-rules do not depend on the current value of the cell). It is easy check that M' accepts $L(G, i, F)$ which is thus context-sensitive.

For the converse implication, let L be context-sensitive language and M a normalized LBM accepting L. The transition graph of M traces L from the initial configuration to an infinite set of configurations (i.e all configurations where the control state is terminal). To restrict accepting configurations to a finite set, we add a new state q_f to the machine M and whenever M makes an ϵ-transition to enter a terminal state, we add the ability to enter configuration $[q_f]$ by a sequence of ϵ-transitions. As $[q_f]$ has no out-going edges, it is an external configuration and it is easy to see that $L = L(G, c_0, [q_f])$. □

Remark 4.55. When a linearly bounded graph is explicitly seen as the transition graph of a LBM, as a Cayley-type graph or as a transduction graph, i.e. when the naming of its vertices is fixed, considering context-sensitive sets of final vertices does not increase the accepted family of languages.

4.3.3.2 Closure properties

Linearly bounded graphs enjoy several good properties, which will be especially important when comparing this class to other families of graphs related to LBMs or context-sensitive languages (see Section 4.4).

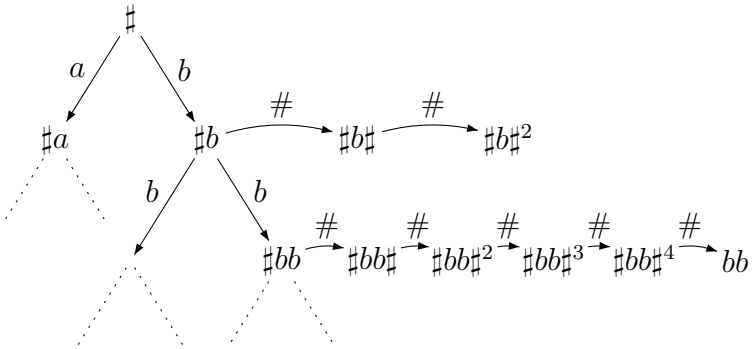


Figure 4.15: The graph G for some L containing bb but not b.

Proposition 4.56. *The family of linearly bounded graphs is closed under restriction to reachable vertices from any vertex and under restriction to a context-sensitive set of vertices.*

Proof. We first prove the closure under restriction to reachable vertices. Let G be a linearly bounded graph given by a family of 1-incremental context-sensitive transductions $(T_a)_{a\in\Sigma}$ and u_0 a vertex in V_G. Each transduction T_a for $a \in \Sigma$ is accepted by a LBM M_a. Consider the graph G' obtained by restricting G to its set of vertices reachable from u_0. Note that this set is in general not context-sensitive, as illustrated by the following example.

Example 4.57. Consider a language $L \subset \{a, b\}^*$ that is recognizable in exponential space but not in linear space (i.e. $L \in \mathrm{NSPACE}[2^n] \setminus \mathrm{NSPACE}[n]$). The existence of such a language is guaranteed by Theorem 1.12. Consider the linearly bounded graph G labeled by $\{a, b, \sharp\}$ and defined by:

- $\sharp u \xrightarrow[G]{x} \sharp ux$ for all $u \in \{a, b\}^*$ and $x \in \{a, b\}$,
- $\sharp u\sharp^n \xrightarrow[G]{\sharp} \sharp u\sharp^{n+1}$ for all $u \in \{a, b\}^*$ and $n + 1 \leq 2^{|u|}$,
- and $\sharp u\sharp^{2^{|u|}} \xrightarrow[G]{\sharp} u$ for all $u \in L$.

Figure 4.15 illustrates the construction of this graph. The set of vertices reachable from $\sharp$ restricted to $\{a, b\}^*$ is precisely the language L which by definition is not context-sensitive.

Therefore, we need to adopt a different naming of the vertices of G'. We construct a graph H isomorphic to G' which is defined by a family of 1-incremental context-sensitive transductions. For any pair of vertices u and $v \in V_G$, we write $u \Rightarrow_n v$ if there exists a path from u to v in G using only vertices of size less than or equal to n. The set of vertices of H is $V_H = \{u\Box^n \mid u \in \Gamma^*, u_0 \Rightarrow_{|u|+n}$

u and $u_0 \not\Rightarrow_{|u|+n-1} u\}$. Note that for all $u \in V_G$, there is a unique n such that $u\Box^n \in V_H$. For all $a \in \Sigma$, we take $u\Box^n \xrightarrow[H]{a} v\Box^m$ iff $u\Box^n \in V_H$, $v\Box^m \in V_H$ and $u \xrightarrow[G]{a} v$.

The graph H is isomorphic to G'. It is easy to see that the mapping $\rho \in V_{G'} \mapsto V_H$ associating to every $u \in V_{G'}$ the unique $u\Box^n \in V_H$ is a bijection and a graph morphism from G' to H. We first prove that V_H is context-sensitive and that for all $a \in \Sigma$, $\xrightarrow[H]{a}$ is a 1-incremental context-sensitive transduction. It will then follow that H and G' are linearly bounded graphs.

1. Consider the language L^+ equal to $\{u\Box^n \mid u_0 \Rightarrow_{|u|+n} u\}$ and the language L^- equal to $\{u\Box^n \mid u_0 \Rightarrow_{|u|+n-1} u\}$, it is easy to check that $V_H = L^+ \cap \overline{L^-}$. If we prove that L^+ and L^- are context-sensitive languages, it follows by Theorem 1.11 that V_H is a context-sensitive language.

   We construct a LBM M accepting L^+. When starting with $u\Box^n$, M guesses a path from u_0 to u with vertices of length at most $|u| + n$. It starts with u_0 and guesses a word u_1 of length at most $|u| + n$ then simulates one of the M_a's to check that $u_0 \xrightarrow[G]{a} u_1$. Finally, M replaces u_0 by u_1 and iterates the process until u_i is equal to u. This can be done using at most $3(|u|+n)$ cells. A similar construction allows to recognize L^-.
2. For all $a \in \Sigma$, $u\Box^n \xrightarrow[H]{a} v\Box^m$ implies that $|v| \leq |u| + 1$ as $u \xrightarrow[G]{a} v$ and $\xrightarrow[G]{a}$ is 1-incremental and $m \leq n$ as $u_0 \Rightarrow_n u$ and $u \xrightarrow[G]{a} v$. Hence, $|v| + m \leq |u| + n + 1$ and therefore $\xrightarrow[H]{a}$ is 1-incremental.

   The language $L_a = \{u\Box^n \sharp v\Box^m \mid u \xrightarrow[H]{a} v\}$ is accepted by a LBM that checks that $u\Box^n$ and $v\Box^m$ belong to V_H (by simulating a machine accepting V_H) and that $u \xrightarrow[G]{a} v$. Hence $\xrightarrow[H]{a}$ is a 1-incremental context-sensitive transduction.

Example 4.58. This construction applied to the graph of Ex. 4.57 gives the graph H defined by:

- $\sharp u \xrightarrow[G]{x} \sharp ux$ for all $u \in \{a, b\}^*$ and $x \in \{a, b\}$,
- $\sharp u \sharp^n \xrightarrow[G]{\sharp} \sharp u \sharp^{n+1}$ for all $u \in \{a, b\}^*$ and $n + 1 \leq 2^{|u|}$,
- and $\sharp u \sharp^{2^{|u|}} \xrightarrow[G]{\sharp} u\Box^{2^{|u|}}$ for all $u \in L$.

Let us now prove the closure of linearly bounded graphs under restriction to a context-sensitive set of vertices. Let G be a linearly bounded graph defined by

a family of $(T_a)_{a\in\Sigma}$ of 1-incremental context-sensitive transduction. By Proposition 4.50, the transduction $T_a \cap \{(u,v) \mid u \in F, v \in F \text{ and} |v| \leq |u|+1\}$ is 1-incremental context-sensitive. It follows that $G|_F$ is a linearly bounded graph. □

Since all rational languages are context-sensitive, linearly bounded graphs are also closed under restriction to a rational set of vertices. This shows that it is not necessary to allow arbitrary rational restrictions in the definition of transition graphs of linearly bounded machine, since such a restriction can be directly applied to the set of external configurations of a machine.

Finally, we can extend the result of [KP99] to the class of linearly bounded graphs, and show that they are closed under synchronized product [AN82].

Proposition 4.59. *The family of linearly bounded graphs is closed under synchronized product, up to isomorphism.*

Proof. The synchronized product of two graphs G and G' with labels in Σ and Σ' with respect to a set of constraints $C \subseteq \Sigma \times \Sigma'$ is the graph

$$G \otimes G' = \{(u,v) \xrightarrow{(a,b)} (u',v') \mid u \xrightarrow[G]{a} u' \wedge v \xrightarrow[G]{b} v' \wedge (a,b) \in C\}.$$

It is straightforward from this definition that if G and G' are both linearly bounded, defined for instance as context-sensitive transduction graphs, the binary edge relations of their synchronized product are also incremental context-sensitive transductions. □

4.3.3.3 Deterministic linearly bounded graphs.

We now consider the relations between the determinism of linearly bounded graphs and the determinism of the linearly bounded machines defining them. As remarked in Section 2.1.1.1, there exist non-deterministic labeled machines, and in particular LBMs, whose transition graphs are deterministic. More precisely, any machine in which, from any given configuration, at most one external configuration is reachable by a partial run labeled by ε^*, has a deterministic transition graph. In fact, we can show that any LBM can be transformed in order to verify this property. Consequently, as expressed by the following proposition, all context-sensitive languages can be accepted by a deterministic linearly bounded graph.

Proposition 4.60. *For all context-sensitive language L, there exists a* deterministic *linearly bounded graph G, a vertex i and a rational set of vertices F of G such that $L = L(G, \{i\}, F)$. Moreover, G can be chosen to be a tree.*

Proof. Let $L \subset \Sigma^*$ be a context-sensitive language. We build a linearly bounded graph G labeled by Σ whose vertices are words of the form ε, Aw or Rw, with $w \in \Sigma^*$ and $A, R \notin \Sigma$ (where A stands for 'accept' and R for 'reject'), and whose edges are defined by the following 1-incremental context-sensitive transductions:

$$\begin{aligned} &Au \xrightarrow{a} Aua, \quad Rv \xrightarrow{a} Ava && \text{for all } u \in L, v \notin L \text{ and } ua, va \in L, \\ &Au \xrightarrow{a} Rua, \quad Rv \xrightarrow{a} Rva && \text{for all } u \in L, v \notin L \text{ and } ua, va \notin L. \end{aligned}$$

The language of this deterministic graph between vertex A if $\varepsilon \in L$, or R if $\varepsilon \notin L$, and the rational set $A\Gamma^*$ is precisely L. Moreover, G is a tree isomorphic to the complete infinite Σ-labeled tree. □

Remark 4.61. The previous result does not stand for a finite set of final vertices even if we only consider deterministic context-sensitive languages. Consider for example, the language $L = \{(a^n b)^* | n \in \mathbb{N}\}$. Suppose that there exists a deterministic graph G such that $L = L(G, i, F)$ for some finite set F. Then there would exists $m \neq n$, such that $i \xrightarrow[G]{a^n b} f$ and $i \xrightarrow[G]{a^m b} f$. As $f \xrightarrow[G]{a^m b} f'$ for some $f' \in F$, it would follow that $i \xrightarrow[G]{a^n b a^m b} f'$.

Of course, we cannot conclude from this that the languages of deterministic linearly bounded graphs are the deterministic context-sensitive languages, which would amount to answering the general question raised by Kuroda [Kur64] whether deterministic and non-deterministic languages coincide. However, if we only consider terminating linearly bounded machines (which have no infinite run on any given input word) the family of transition graphs we obtain faithfully illustrates the determinism of the languages.

Remark 4.62. Even for terminating LBMs, the determinism of a transition graph does not imply that the machine defining it is deterministic. Figure 4.16 illustrates this fact. The idea of the following proof is to show that any terminating LBM whose transition graph is deterministic can be determinized without changing the structure of the graph.

Proposition 4.63. *The languages of deterministic transition graphs of terminating linearly bounded machines (between an initial vertex and a rational set of final vertices) are the deterministic context-sensitive languages.*

Proof. Let L be a deterministic context-sensitive language. There exists a deterministic terminating LBM M accepting L. By definition, its transition graph G_M is deterministic, and it accepts L between the vertices corresponding to the initial configuration and set of final configurations of M.

Conversely, let G_M be the deterministic transition graph of a terminating LBM $M = (Q, \Sigma, \Gamma, \delta, q_0, F, [,])$. We construct a deterministic LBM N whose

Figure 4.16: Configuration graph and deterministic transition graph of a non-deterministic terminating labeled LBM

---

transition graph G_N is equal to G_M. The machine N is obtained from M by keeping exactly one rule in δ with a given left-hand side and label. The machine N is equal to $(Q, \Sigma, \Gamma, \delta', q_0, F, [,])$ where δ' is a subset of δ:

- if $pA \xrightarrow{x} qU \in \delta$ then there exists $pA \xrightarrow{x} q'U' \in \delta'$,
- if $pA \xrightarrow{x} qU \in \delta'$ then for all $pA \xrightarrow{x} q'U' \in \delta'$, $q = q'$ and $U = U'$.

By construction, N is deterministic. Moreover, the set of external configurations of N is equal to the set of external configurations of M. It remains to prove that for any two external configurations c and c', $c \xrightarrow[G_M]{a} c'$ if and only if $c \xrightarrow[G_N]{a} c'$.

If $c \xrightarrow[G_M]{a} c'$, there exists a path between c and c' labeled by $a\varepsilon^*$ in the configuration graph C_M of M. As G_M is deterministic and terminating, any maximal path in C_M labeled by $a\varepsilon^*$ ends in c'. This property still holds for the configuration graphs of C_N. Suppose by contradiction that there exists a maximal path in C_N labeled by $a\varepsilon^*$ starting from c and ending in $c'' \neq c'$. This implies the c'' has out-going ϵ-edges in C_M and not in C_N which is impossible by construction of N.

Conversely if $c \xrightarrow[G_N]{a} c'$ then, by construction of N, we have $c \xrightarrow[G_M]{a} c'$. Hence, it follows that $G_M = G_N$. □

Remark 4.64. Other equivalent definitions of deterministic transition graphs of terminating labeled LBMs can be given: they coincide with the Cayley-type graphs of strongly normalizing length-decreasing rewriting systems, and to the graphs of deterministic terminating incremental context-sensitive transductions.

Remark 4.65. One can not use the previous constructions to determinize any LBM. Indeed, the construction from Proposition 4.60 produces non terminating machines in general, and the construction from Proposition 1.10 does not preserve the structure of a machine's transition graph.

4.3.3.4 Logical properties

To conclude this section on properties of linearly bounded graphs, we investigate the decidability of the first-order theory of configuration graphs of LBMs and linearly bounded graphs. Due to the high expressive power of the model considered, only local properties expressed in first-order logic can be checked on LBM configuration graphs.

Proposition 4.66. *The family of configuration graphs of linearly bounded Turing machines has a decidable first-order theory.*

Proof. This is a direct consequence of the fact that configuration graphs of LBMs are synchronized rational graphs, which have a decidable first-order theory [BG00]. □

However, as remarked by [KP99], there exists no algorithm which, given a LBM M and a first-order sentence ϕ, decides whether the transition graph of M satisfies ϕ. This statement can be strengthened to the following proposition.

Proposition 4.67. *There exists a linearly bounded graph with an undecidable first-order theory.*

Proof. Consider a fixed enumeration $(M_n)_{n\in\mathbb{N}}$ of all (unlabeled) Turing machines, and a labeled Turing machine M whose language is the set of all words of the form $\#^n$ such that machine M_n halts on the empty input. Using only ε-transitions, M guesses a number n and writes $\#^n$ on its tape, then simulates machine number n on the empty input. If the machine halts, then M reads word $\#^n$ and stops. All accepting runs of M are thus labeled by $\varepsilon^*\#^n$ for some n.

When replacing ε by an observable symbol τ, the configuration graph C of M is a linearly bounded graph. Let C' be its restriction to vertices reachable from the initial configuration of M. The graph C' is still linearly bounded by Proposition 4.56. For all n, the formula ϕ_n expressing the existence of a path labeled by $\tau\#^n$ ending in a vertex with no successor is satisfied in C' if and only if machine M_n halts on input ε. The set of all such satisfiable formulas is not recursive. Hence the first-order theory of C' is undecidable (not even recursively enumerable). □

4.4 Comparison of Both Families

We will now give some remarks about the comparison between linearly bounded graphs and several different sub-families of rational graphs. First note that since linearly bounded graphs have by definition a finite degree, it is therefore only relevant to consider rational graphs of finite degree. However, even under this

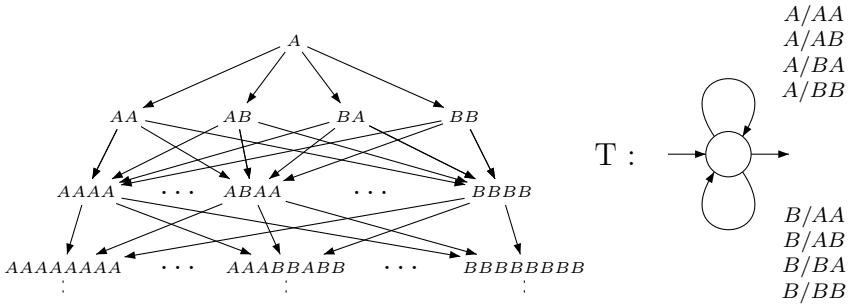


Figure 4.17: A finite degree rational graph (together with its transducer) which is not isomorphic to any linearly bounded graph.

---

structural restriction, rational and linearly-bounded graphs are incomparable, due to the incompatibility in the growth rate of their vertices' degrees.

A first observation is that in a rational graph the out-degree at distance n from any vertex can be c^{c^n} for some constant c (cf. Proposition 2.9), whereas in a linearly bounded graph it is at most c^n.

Lemma 4.68. *For any linearly bounded graph G and any vertex x, there exists $c \in \mathbb{N}$ such that the out-degree of G at distance $n > 0$ of x is at most c^n.*

Proof. Let $(T_a)_{a\in\Sigma}$ be a set of incremental context-sensitive transductions describing G and $k_a \in \mathbb{N}$ such that T_a is a k_a-incremental context-sensitive transduction. We take k to be the maximum of $\{k_a \mid a \in \Sigma\} \cup \{|x|\}$. At distance $n > 0$ of x, a vertex has length at most $k(n+1)$ and hence the out-degree is bounded by the number of vertices of length at most $k(n+2)$ which is less than $|\Gamma|^{k(n+2)+1}$. Hence, there exists $c \in \mathbb{N}$ such that the out-degree is bounded by c^n. □

Figure 4.17 shows a rational graph whose vertices at distance n from the root A have out-degree $2^{2^{n+1}}$. This graph is thus not linearly bounded.

Conversely, in a rational graph of finite degree, the in-degree at distance n from any vertex is at most c^{c^n} for some $c \in \mathbb{N}$. In a linearly bounded graph it can be as large as $f(n)$ for any mapping f from $\mathbb{N}$ to $\mathbb{N}$ recognizable in linear space (i.e. such that the language $\{0^n 1^{f(n)} \mid n \in \mathbb{N}\}$ is context-sensitive).

Lemma 4.69. *For any mapping $f : \mathbb{N} \mapsto \mathbb{N}$ recognizable in linear space, there exists a linearly bounded graph G with a vertex x such that the in-degree of any vertex at distance $n > 0$ of x can be as large as $f(n)$.*

Proof. Let f be a mapping from $\mathbb{N}$ to $\mathbb{N}$ recognizable in linear space, and let G_f be the linearly bounded graph defined using the following incremental context-sensitive transduction:

$$\begin{aligned} T \;=\; & \{(u, u0) \mid u \in 0^*\} \;\cup\; \{(u, u1) \mid u \in 0^n 1^m,\; m < f(n)\} \\ & \cup \{(uv, u) \mid u \in 0^*, v \in 1^* \text{ and } |v| \leq f(|u|)\} \end{aligned}$$

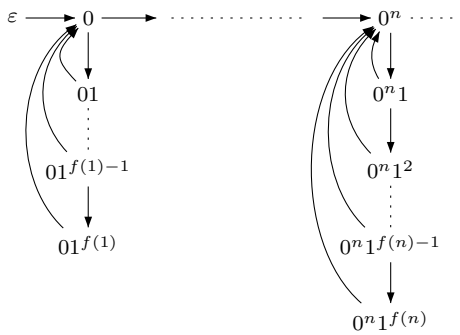


Figure 4.18: A linearly bounded graph which is not isomorphic to any rational graph.

Figure 4.18 illustrates the construction of G_f. The in-degree of vertex 0^n at distance n from the root ε is equal to $f(n)$, for any mapping f recognizable in linear space. It is quite straightforward to build a LBM recognizing the language $\{0^n1^m \mid m \leq f(n)\}$ for $f(n) = 2^{2^{2^n}}$ for instance, which immediately implies that G_f is linearly bounded and of finite degree, but not rational. □

An instance of such a mapping is $f : n \mapsto 2^{2^{2^n}}$, which is more than the in-degree at distance n of a vertex in any rational graph of finite degree. From these two observations, we easily get:

Proposition 4.70. *The families of finite degree rational graphs and of linearly bounded graphs are incomparable.*

Since finite-degree rational graphs and linearly bounded graphs are incomparable, we investigate more restricted sub-families of rational graphs. For synchronized graphs of finite out-degree, we have the following result:

Proposition 4.71. *The synchronized graphs of finite out-degree form a strict sub-family of linearly bounded graphs.*

Proof. By Proposition 4.49, all synchronized relations of finite image are incremental. Hence, all synchronized graphs of finite out-degree are linearly bounded graphs. The strictness can be deduced from the fact that synchronized graphs are not closed under restriction to reachable vertices from a given vertex (cf Proposition 4.56). □

For the even more restricted family of bounded-degree rational graphs, we show the following comparison:

Theorem 4.72. *The rational graphs of bounded degree form a sub-family of linearly bounded graphs of bounded degree (up to isomorphism).*

Proof. The inclusion result is based on the previously used uniformization result by Weber 4.35. Let $G = (T_a)_{a\in\Sigma}$ be a rational graph over Γ whose out-degree is bounded by k. Each relation T_a is therefore k-valued and hence there exist k rational functions $F_{a_1}, \dots F_{a_k}$ such that $T_a = \bigcup_{i\in[1,k]} F_{a_i}$. We write $X = \{a_i \mid a \in \Sigma \text{ and } i \in [1,k]\}$.

We are going to represent each vertex x of G by a pair $(w,t) \in V_G \times X^*$ such that $x = F_{t_{|t|}}(\dots(F_{t_1}(w)))$. However, there can be several such pairs for a given vertex. We are going to define a total order $<$ on $V_G \times X^*$ and associate to x the smallest such pair. First, let $<_\Gamma$ and $<_X$ be two arbitrary total orders over Γ and X respectively and let $\prec_\Gamma$ and $\prec_X$ be the lexicographic orders induced by $<_\Gamma$ and $<_X$ respectively.

$$\begin{array}{lll} (m,t) < (n,r) & \text{iff} & |m| + |t| < |n| + |r| \\ & \text{or} & |m| + |t| = |n| + |r| \text{ and } m \prec_\Gamma n \\ & \text{or} & |m| + |t| = |n| + |r|, m = n \text{ and } t \prec_X r \end{array}$$

It is straightforward to prove that $<$ is a total order on $V_G \times X^*$. We now prove that the function N associating to any pair (w,t) the smallest pair (m,r) such that $F_t(w)$=$F_r(m)$. is a 0-incremental context-sensitive transduction.

There are two important points in this proof. The first one is to be able to check in linear space that for any two given pairs (m,r) and (w,t), $F_r(m) = F_t(w)$. In other terms, we want to prove that the language $L = \{m\sharp r\$t\sharp w \mid F_r(m) = F_t(w)\}$ is context-sensitive.
Let $\bar{X}$ be a finite alphabet disjoint from but in bijection with X (for all $x \in X$, we write $\bar{x}$ the corresponding symbol in $\bar{X}$) , we consider the rational graph $\bar{G}$ defined by the family of transducers $(F_a)_{a\in X} \cup (\bar{F}_{\bar{a}})_{a\in X}$ where for all $\bar{a} \in \bar{X}$, $\bar{F}_{\bar{a}} = F_a^{-1}$. It is easy to check that for all $m, w \in \Gamma^*$ and all $r, t \in X^*$, there exists a path $m \xrightarrow[\bar{G}]{r\bar{t}} w$ iff $F_r(m) = F_t(w)$ where $\bar{t} = \bar{t}_{|t|} \dots \bar{t}_1$. By Corollary 4.19, the language $\{i\sharp w\sharp f \mid w \in L(\bar{G}, i, f)\} \cap \Gamma^*\sharp X^*\bar{X}^*\sharp\Gamma^*$ is a context-sensitive language. It follows that L is also context-sensitive.

The second point is to perform this test for all pairs $(m,r) < (w,t)$. Even though the number of such pairs is finite, there is a problem when the testing fails. Indeed, this may mean that $m\sharp r\$t\sharp w \notin L$, or that the path actually exists and that there exists another successful computation. Hence, on a failed test we cannot simply move to the next (m,r) candidate. To deal with this problem, an idea is to perform a test of membership in the complement $\overline{L}$ of L, which is a context-sensitive language too. In case the complement test fails, this computation of $N(w,t)$ can safely fail, since we can assure that another successful computation for either the membership test or its complement exists. In case the

complement test succeeds, it means the current pair (m, r) is not valid and we can proceed to the following one. The computation of $N(w, t)$ succeeds when a suitable (m, r) pair is found, and fails when all pairs $(m, r) < (w, t)$ have been tested and rejected. N is 0-incremental since by definition of the total order $<$, $(m, r) < (w, t) \implies |m| + |r| \leq |w| + |t|$, and all steps of the construction can be done in linear space.

We can now define the linearly bounded graph L with vertices in $\Gamma^* \# X^*$ defined by the set of transductions $(T_a)_{a \in \Sigma}$ defined as follows:

$$T_a = \bigcup_{i \in [1,k]} \{(w\#x, w'\#y) \mid N((w, x)) = (w, x) \text{ and } N(w, xa_i) = (w', y)\}.$$

As N is 0-incremental, then T_a is 1-incremental. The mapping ρ from V_G to $V_G \times X^*$ associating to a vertex $x \in V_G$ the smallest (w, t) such that $x = F_t(w)$ is a bijection from V_G to V_L, which induces an isomorphism from G to L. Hence, G is a linearly bounded graph, which concludes the proof of inclusion. □

Example 4.73. Let Γ be a singleton alphabet, and consider the pair of transductions $T_a = \{(n, 2n) | n \geq 1\}$ and $T_b = \{(n, n-3) | n \geq 4\}$ on the domain of words over Γ seen as unary-coded positive integers. Let us apply the construction from the previous inclusion proof to the bounded-degree rational graph G defined by T_a and T_b. Here $<_\Gamma$ is trivial, and we take $a <_X b$. The graph G and the linearly bounded graph G' obtained by applying the construction are shown on Fig. 4.19. Note for instance how vertex 13 in G is represented by vertex $(2, a^3b)$ in G' because $(2, a^3b)$ is the smallest pair (u, v) such that $u \xrightarrow[G]{v} 13$.

In fact, we can prove that the previous inclusion is strict.

Theorem 4.74. *There exists a linearly bounded graph of bounded degree which is not a rational graph.*

Proof. We now prove that there exist a linearly bounded graph of bounded degree which is not isomorphic to any bounded degree rational graph. Let us first establish that linearly bounded graphs of bounded degree are not closed under edge reversal. Let $L \subseteq \{0, 1\}^+$ be a language in NSPACE$[2^n] \setminus$ NSPACE$[n]$ (cf Theorem 1.12), we define a linearly bounded graph G with vertices in $0\{0, 1\}^+\#^* \cup \bar{0}\{\bar{0}, \bar{1}\}^+$ and edges defined by:

$$\xrightarrow{x} = \{(w, wx) \mid w \in 0\{0, 1\}^*\} \cup \{(w, w\bar{x}) \mid w \in \bar{0}\{\bar{0}, \bar{1}\}^*\} \qquad \text{for } x \in \{0, 1\}$$

$$\xrightarrow{\#} = \{(w, w\#) \mid w = uv, u \in 0\{0, 1\}^*, v \in \#^* \text{ and } |v| < 2^{|u|}\}$$
$$\cup \{(w, \bar{u}) \mid w = uv, u \in 0\{0, 1\}^*, v \in \#^*, |v| = 2^{|u|} \text{ and } u \in L\}$$

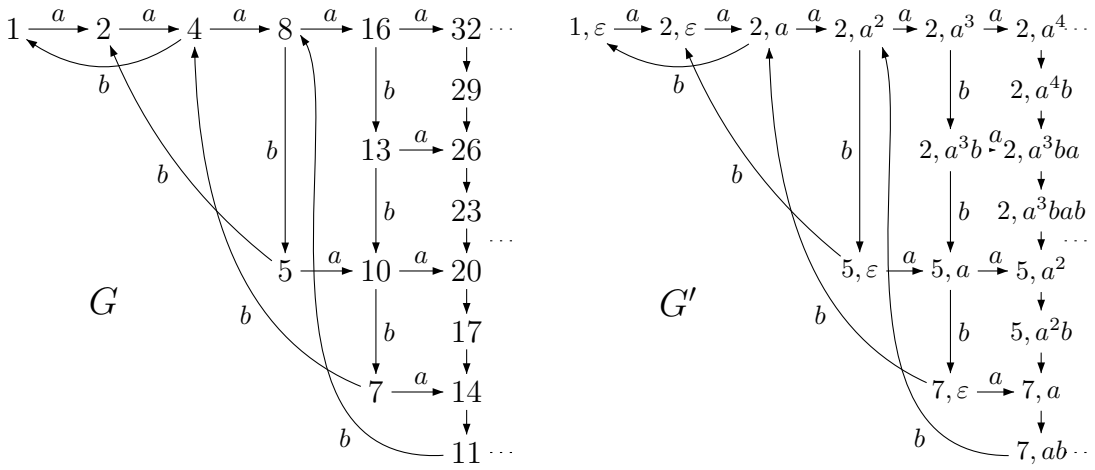


Figure 4.19: Bounded-degree rational graph G and isomorphic linearly bounded graph G'.

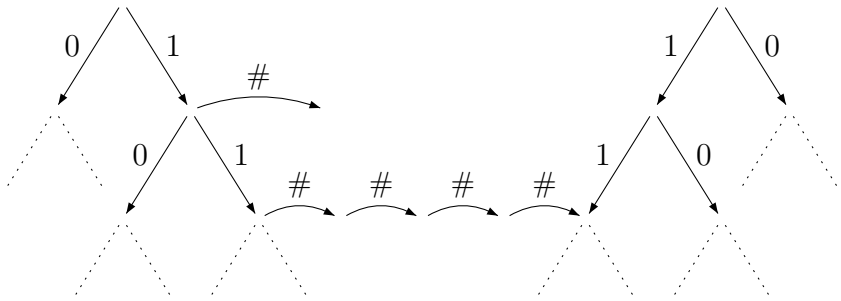


Figure 4.20: The graph G for some L containing 11.

The relations $\xrightarrow{0}$, $\xrightarrow{1}$ and $\xrightarrow{\#}$ are incremental context-sensitive transductions. In fact, as L belongs to NSPACE[2^n], the language $\{w\#^{2^{|w|}} \mid w \in L\}$ belongs to NSPACE[n]. The construction of G_L is illustrated by Figure 4.20.

Suppose that linearly bounded graphs of bounded degree were closed under edge reversal. It would follow that H obtained from G by reversing the edges labeled by # is also a linearly bounded graph. As H is linearly bounded, we can assume that $\xrightarrow[H]{0}$, $\xrightarrow[H]{1}$ and $\xrightarrow[H]{\#}$ are incremental context-sensitive transductions. Let x be the vertex of H corresponding to $\bar{0}$. The set of vertices $F = \mathrm{Dom}(\xrightarrow[H]{\#})$ is a context-sensitive language. It follows from Proposition 4.54 that $L(H, \{x\}, F)$ is context-sensitive. As $L = L(H, \{x\}, F) \cap \{0, 1\}^*$, L would also be context-sensitive which contradicts its definition.

As rational graphs are closed under edge reversal, it follows from Theorem 4.72 that rational graphs of bounded degree are strictly contained in linearly bounded graphs of bounded degree. □

It may be interesting at this point to recall that there are strong reasons to believe that the languages accepted by finite degree synchronized graphs are strictly included in context-sensitive languages (cf. Theorem 4.32). Furthermore, all existing proofs that the rational graphs accept the context-sensitive languages break down when the out-degree is bounded. It is thus not clear whether rational graphs of bounded degree accept all context-sensitive languages. However, as noted in Proposition 4.60, it is still the case for bounded degree linearly bounded graphs, and in particular for deterministic linearly bounded graphs.

4.5 Another Chomsky-like Hierarchy of Graphs

Our structural observations concerning the degree and number of initial vertices of infinite graphs seen as automata lead us to consider a more restricted notion of infinite automata than that provided by the families of graphs summarized in Section 4.1, closer to classical finite automata (as was already observed in [Car05]), and to propose a hierarchy of families of infinite graphs of bounded degree with a single distinguished initial vertex accepting the families of languages of the Chomsky hierarchy, in the spirit of [CK02a].

Finite graphs obviously have a bounded degree, and they accept rational languages. The transition graphs of real-time pushdown automata, which accept all context-free languages, are the pushdown graphs [MS85], or equivalently the HR graphs of bounded degree [Cou89] and prefix-recognizable graphs of bounded degree [CK01]. By Proposition 4.60, the languages of deterministic linearly bounded

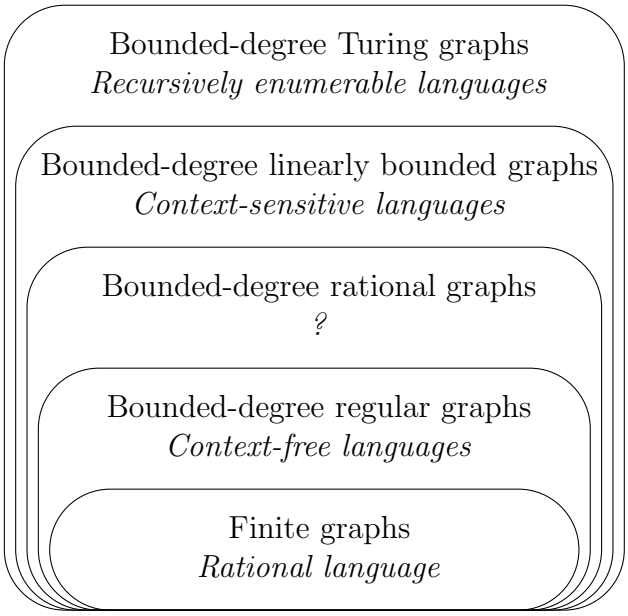


Figure 4.21: A Chomsky-like hierarchy of bounded-degree infinite graphs.

graphs are the context-sensitive languages. Deterministic graphs have by definition a bounded degree, so bounded degree linearly bounded graphs also accept the same family of languages. Note that, even though rational graphs of bounded degree are a strict subset of linearly bounded graphs of bounded degree (Theorem 4.72), it is still unknown whether they accept all context-sensitive languages. Finally, since deterministic Turing machines have bounded-degree transition graphs and accept all recursively enumerable languages, we can also restrict ourselves to the family of bounded degree Turing graphs. These families are summarized on Figure 4.21.

Note that all these families of graphs can be characterized as transition graphs, and that the families of machines used to define them form a strict hierarchy.

Chapter 5

Reachability Analysis of Higher-Order Context-Free Processes

Higher-order pushdown systems are a powerful modeling tool for higher-order functional programs, i.e. programs whose functions and procedures accept functions as parameters and may produce functions as results. For instance, a translation of higher-order program schemes into higher-order pushdown automata and conversely can be found in [KNU02]. The definition of these automata can be found in Section 1.3.4. Moreover, we know from recent results in the field of infinite graphs that the monadic second-order theory of their transition graphs are decidable (cf. Section 2.2.1.5). In terms of verification, this means that any property expressible in MSO or weaker logics over graphs can be checked against a given higher-order pushdown automaton transition graph.

In this chapter, we are interested in providing an extension to higher-order pushdown systems of the direct symbolic reachability analysis algorithm presented in [BEM97]. The symbolic representation of infinite sets of configurations chosen in this algorithm is by regular languages.

5.1 Higher-order Context-free Processes

We introduce a class of models we call *higher-order context-free processes*, which generalize context-free processes (CFP) and are a subclass of higher-order pushdown automata (cf. Section 1.3.4). First, we define a new set of *level 1 rewriting* operations $\{push_1^w \mid w \in \Gamma^*\}$ over higher-order stacks, each of which is equivalent to a pop_1 operation followed by a sequence of $push_1$ operations. Formally, let $w = a_1 \dots a_k \in \Gamma^*$ be a sequence of stack symbols, we define $push_1^w$ as

$pop_1 \circ push_1^{a_k} \circ \ldots \circ push_1^{a_1}$. In particular, notice that $push_1^{\varepsilon} = pop_1$, and that, when the top-most symbol of a stack s is a, $push_1^b a(s) = push_1^b(s)$.

Using these new operations, we now define a sub-family of higher-order pushdown automata, which coincides at level 1 with the so-called *context-free processes*[1] [BK89].

Définition 5.1. *A* higher-order context-free process *of level n (or n-HCFP) is a pair $H = (\Gamma, \delta)$, where Γ is a finite alphabet and δ is a finite set of transitions of the form (a, o), where $a \in \Gamma$ and $o \in \{push_1^w \mid w \in \Gamma^*\} \cup \{push_n, pop_n \mid n > 1\}$.*

Since we are not interested in languages accepted by this class of automata, but rather by the relation they induce between their configurations, we only consider unlabeled transition rules. A configuration of a n-HCFP H is a n-stack over Γ. H defines a transition relation $\underset{H}{\longrightarrow}$ between n-stacks (or $\longrightarrow$ when H is clear from the context), where

$$s \underset{H}{\longrightarrow} s' \iff \exists (a, o) \in \delta \text{ such that } top(s) = a \text{ and } s' = o(s).$$

Since $push_1$ rules are defined using ordinary higher-order pushdown operations, higher-order context-free processes can be seen as higher-order pushdown automata with a single control state where level 1 operations may be chained, the same way context-free processes can be seen as pushdown automata with a single control state.

The level $\mathrm{l}(d)$ of a transition $d = (a, o)$ is simply the level of o. Let us give a few more notations concerning HCFP computations. Let $H = (\Gamma, \delta)$ be a n-HCFP. A *run* of H starting from some stack s_0 is a sequence $s_0 s_1 \ldots s_k$ such that for all $i \in [1, k]$, $s_{i-1} \longrightarrow s_i$. The reflexive and transitive closure of $\longrightarrow$ is written $\overset{*}{\longrightarrow}$ and called the *reachability* relation. For a given set C of n-stacks, we also define the *constrained transition* relation $\longrightarrow_C = \longrightarrow \cap (C \times C)$, and its reflexive and transitive closure $\overset{*}{\longrightarrow}_C$. For any set of n-stacks S, we consider the sets:

$$\begin{aligned}
post_H[C](S) &= \{\, s \mid \exists s' \in S,\ s' \longrightarrow_C s \,\},\\
post_H^*[C](S) &= \{\, s \mid \exists s' \in S,\ s' \overset{*}{\longrightarrow}_C s \,\},\\
pre_H[C](S) &= \{\, s \mid \exists s' \in S,\ s \longrightarrow_C s' \,\},\\
pre_H^*[C](S) &= \{\, s \mid \exists s' \in S,\ s \overset{*}{\longrightarrow}_C s' \,\}.
\end{aligned}$$

When C is the set $\S_n$ of all n-stacks, we omit it in notations and simply write for instance $pre_H(S)$ instead of $pre_H[C](S)$. We will also omit H when it is clear from the context. When H consists of a single transition d, we may write $pre_d(S)$ instead of $pre_H(S)$.

[1] Also called *basic process algebras* in the field of process algebras.

5.2 Sets of Stacks and Symbolic Representation

To be able to design symbolic verification techniques over higher-order context-free processes, we need a way to finitely represent infinite sets (or languages) of configurations. In this section we present the sets of configurations (i.e. sets of stacks) we consider, as well as the family of automata which recognize them.

A n-stack $s = [s_1 \dots s_l]$ over Γ is associated to a word $w(s) = [w(s_1) \dots w(s_l)]$, in which stack letters in Γ only appear at nesting depth n (we will often make no distinction between a stack s and its associated word $w(s)$). A set of stacks over Γ is called *regular* if its set of associated words is accepted by a finite automaton over $\Gamma' = \Gamma \cup \{ [\,,]\,\}$, which in this case we call a *stack automaton*.

Let A be such an automaton. A state p of A is of level 0 if it has no successor by $[$ and no predecessor by $]$. It is of level k if all its successors by $[$ and predecessors by $]$ are of level $k-1$. The level of p is written $\mathrm{l}(p)$. We also define a notion of level for paths. A *level n path* in a stack automaton is a path $p_1 \dots p_k$ with $\mathrm{l}(p_1) = \mathrm{l}(p_k) = n$ and $\forall i \in [2, k-1],\ \mathrm{l}(p_i) < n$. All such paths are labeled by n-stacks. Finally, to concisely refer to the whole set of level n paths between two level n control states, we introduce the following notation. Let

$$Q = \{\, q \in Q_A \mid \mathrm{l}(q) < n \wedge p_1 \longleftrightarrow A{+}q \longleftrightarrow A{+}p_2 \,\}$$

be the set of all states of A occurring on a level n path between p_1 and p_2. If Q is not empty, we write $p_1 \overset{B}{\underset{A}{\rightsquigarrow}} p_2$, where B is defined as:

$$B = \big(\, Q_B = Q \cup \{p_1, p_2\},\ \Gamma',\ \delta_B = \delta_A \cap (Q_B \times \Gamma' \times Q_B),\ p_1,\ p_2 \,\big).$$

This notation allows in a way to isolate ¡¡ sub-automata ¿¿ of lesser level inside stack automata of higher level. Following this idea, and due to the nested structure of pushdown stacks, it will sometimes be more convenient to characterize sets of stacks using *nested stack automata*.

Définition 5.2. *A level* 1 *nested stack automaton is a finite automaton whose transitions have labels in* Γ. *A nested stack automaton of level* $n \geq 2$ *is a finite automaton whose transitions are labeled by level* $n-1$ *nested automata over* Γ.

Let $A = (Q, \Gamma, \delta, q_0, q_f)$ be a level n nested automaton[2] with $n \geq 2$. The existence of a transition labeled by automaton B between two control states p and q in A is written $p \longleftrightarrow ABq$, or simply $p \xrightarrow{B} q$ when A is clear from the context. The level k language of A for $k \in [1, n]$ is defined recursively as:

$$\begin{aligned} L_k(A) &= \{\, [L_k(A_1) \dots L_k(A_l)] \mid [A_1 \dots A_l] \in L_n(A) \,\} && \text{if } k < n, \\ L_k(A) &= \{\, [A_1 \dots A_l] \mid q_0 \longleftrightarrow AA_1 \dots \longleftrightarrow AA_l q_f \,\} && \text{if } k = n. \end{aligned}$$

[2]Note that we only consider automata with a single final state.

For simplicity, we often abbreviate $L_1(A)$ as $L(A)$. We say a nested automaton B occurs in A if B labels a transition of A, or occurs in the label of one. Level n automata are well suited for representing sets of n-stacks, but have the same expressive power as standard level 1 stack automata.

Proposition 5.1. *The stack languages accepted by nested stack automata are the regular stack languages.*

Proof. Let $A = (Q, \Gamma, \delta, i, f)$ be a nested stack automaton, we compute by induction on the level n of A a level 1 stack automaton $\flat A$ such that $L_1(A) = L(\flat A)$. For $n = 1$, $A = \flat A$. For greater values of n, let $A_1 \dots A_m$ be the level $n-1$ automata labeling the transitions of A. By induction hypothesis, we can build level 1 automata $\flat A_1 \dots \flat A_m$ such that $\forall j \in [1, m]$, $L_1(A_j) = L(\flat A_j)$. Let $\flat A_j = (Q_j, \Gamma, \delta_j, i_j, f_j)$, with all Q_j supposed disjoint. We now build the level 1 automaton $\flat A = (Q', \Gamma, \delta', i', f')$ where for all $p, q \in Q$, $j \in [1, m]$, $r, s, t, u \in Q_j$ and $a \in \Gamma'$ such that $p \longleftrightarrow A A_j q$, $i_j \longleftrightarrow A_j \downarrow [r$, $s \longleftrightarrow A_j \downarrow a t$ and $u \longleftrightarrow A_j \downarrow] f_j$, we have:

$$i' \longleftrightarrow A\downarrow[i \quad p \longleftrightarrow A\downarrow[pr \quad ps \longleftrightarrow A\downarrow apt \quad pu \longleftrightarrow A\downarrow]q \quad f \longleftrightarrow A\downarrow]f'$$

According to this construction, a path of $\flat A$ between two control states p and q in $Q \cap Q'$ is labeled by a word s if and only if s represents a $(n-1)$-stack accepted by some A_j such that $p \longleftrightarrow A A_j q$. Hence $\flat A$ accepts all words of the form $[s_1 \dots s_l]$ such that $[A_{i_1} \dots A_{i_l}] \in L_n(A)$ and for all j, $s_j \in L(A_{i_j})$, which is precisely the definition of $L(A)$.

Conversely, let $A = (Q, \Gamma', \delta, i, f)$ be a level 1 automaton recognizing n-stacks. We want to build a level n nested automaton $A' = (Q', \Gamma, \delta', i', f')$ such that $L_1(A') = L(A)$. As no path of A labeled by a word which does not denote a correct stack can be accepting, we may consider without loss of generality that the level of every state in Q is well-defined. Let Q_{n-1} be the set of level $n-1$ states of A. The only states of level n are i and f. If $n = 1$, we build A' with a set of states $Q' = Q_{n-1}$ and the following set of transitions:

$$\delta' = \{\, i' \xrightarrow{a} q \mid i \longleftrightarrow A[p \longleftrightarrow Aaq \,\} \cup \big(\delta \cap (Q_{n-1} \times \Gamma \times Q_{n-1})\big) \cup \{\, p \xrightarrow{a} f' \mid p \longleftrightarrow Aaq \longleftrightarrow A]f \,\}.$$

If $n > 1$, for each $p, q \in Q_{n-1}$ and B such that $p \overset{B}{\rightsquigarrow} q$, we first build inductively a nested automaton B' such that $L_1(B') = L(B)$. We then give A' the following

set of transitions:

$$\delta' = \{\, i' \xrightarrow{B'} q \mid \exists p, q \in Q_{n-1},\ i \longleftrightarrow A[p \overset{B}{\underset{A}{\rightsquigarrow}} q \,\}$$
$$\cup\ \{\, p \xrightarrow{B'} q \mid \exists p, q \in Q_{n-1},\ p \overset{B}{\underset{A}{\rightsquigarrow}} q \,\}$$
$$\cup\ \{\, p \xrightarrow{a} f' \mid \exists p, q \in Q_{n-1},\ p \overset{a}{\underset{A}{\rightsquigarrow}} q \longleftrightarrow A]f \,\}.$$

A stack s is accepted by A' if and only if there is a path in A' labeled by $B'_1 \dots B'_k$ from i' to f' such that $s \in [L_1(B'_1) \dots L_1(B'_k)]$. We thus also have $s \in [L(B_1) \dots L(B_k)]$, and hence $s \in L(A)$. □

Moreover, regular n-stack languages are closed under union, intersection and complement in $\S_n$. We define for later use the set of automata $\{\, A^n_a \mid a \in \Gamma,\ n \in \mathbb{N} \,\}$ such that for all a and n, $L(A^n_a) = \{\, s \in \S_n \mid top(s) = a \,\}$. We also write $A \times B$ the classical product operation over automata such that $L(A \times B) = L(A) \cap L(B)$.

Before proceeding, we have to present a few additional definitions and notations. To be able to easily express and manipulate sets of possible runs of nested automata, we first define the notion of *stack expression*.

Définition 5.3. *A stack expression of level* 0 *over alphabet* Γ *is simply a letter in* Γ. *A stack expression of level* $n > 0$ *is either a* n*-stack* s*, the name* A *of a (nested or not)* n*-stack automaton, a concatenation of level* n *stack expressions, a level* $n-1$ *stack expression between square brackets* $[e]$*, or the repeated concatenation* e^+ *of a level* n *expression* e.

Also, to describe runs of nested automata we define a binary relation $\longmapsto$, which expresses the choice of a particular path in a nested automaton appearing inside a stack expression.

Définition 5.4. *Let* $e = uAv$ *be a stack expression where* A *is a nested* n*-stack automaton, we write* $e \longmapsto u[w]v$ *whenever* $w \in L_n(A)$. *As usual, we write* $\overset{*}{\longmapsto}$ *the reflexive and transitive closure of* $\longmapsto$. *A sequence of stack expressions* $e_1 \dots e_m$ *such that* $e_1 = A$, $e_m \in \S_n$ *and* $\forall i \in [1, m-1]$, $e_i \longmapsto e_{i+1}$ *is called a run of* A.

Finally, we define a concatenation operation over stacks and stack expressions.

Définition 5.5. *Let* $e = [e_1 e_2]$, f *and* g *be stack expressions, we write* $e = f \cdot g$ *if either* $f = e_1$ *and* $g = [e_2]$, *or* $e_1 = f \cdot g'$ *and* $g = [g' e_2]$. *Note that if* e *is a letter in* Γ *or an automaton, there are no* f *and* g *such that* $e = f \cdot g$.

For instance, we could write $[[aB][a][bcd]] = a \cdot [[B][a][bcd]]$, or $[[aB][a][bcd]] = [aB][a] \cdot [[bcd]]$.

5.3 Symbolic Reachability Analysis

Our goal in this section is to investigate effective techniques to compute the sets $pre(S)$, $post(S)$, $pre^*(S)$ and $post^*(S)$ for a given n-HCFP H, in the case where S is a regular set of stacks. For level 1 pushdown systems, it is a well-known result that both $pre^*_H(S)$ and $post^*_H(S)$ are regular. We will see that this is still the case for $pre(S)$ and $pre^*(S)$ in the higher-order case, but not for $post(S)$ (hence not for $post^*(S)$ either).

5.3.1 Forward Reachability

Proposition 5.2. *Given a n-HCFP H and a regular set of n-stacks S, the set $post(S)$ is in general not regular. This set is a context-sensitive language.*

Proof. Let $post_{(a,o)}(S)$ denote the set $\{\, s' \mid \exists s \in S,\ top(s) = a \ \wedge\ s' = o(s) \,\}$. Suppose S is a regular set of n-stacks, then if $d = (a, push_1^w)$ or $d = (a, pop_k)$, it is not difficult to see that $post_{(a,o)}(S)$ is regular. However, if $d = (a, push_k)$ with $k > 1$, then $post_{(a,o)}(S)$ is the set $\{\, [^{n-k+1} t\, t\, w \mid [^{n-k+1} t\, w \in S \,\}$. It can be shown using the usual pumping arguments that this set is not regular, because of the duplication of t. However, one can straightforwardly build a linearly bounded Turing machine recognizing this set. □

5.3.2 Backward Reachability

We first propose a transformation on automata which corresponds to the *pre* operation on their language. In a second time, we extend this construction to deal with the more difficult computation of pre^* sets.

Proposition 5.3. *Given a n-HCFP H and a regular set of n-stacks S, the set $pre(S)$ is regular and effectively computable.*

We introduce a construction which, for a given HCFP transition d and a given regular set of n-stacks S recognized by a level n nested automaton A, allows us to compute a nested automaton A'_d recognizing the set $pre(S)$ of direct predecessors of S by d. This construction is a transformation over nested automata, which we call T_d. We define $A'_d = T_d(A) = (Q', \Gamma, \delta', q'_0, q_f)$ as follows.
If $\mathrm{l}(d) < n$, we propagate the transformation to the first level $n-1$ automaton encountered along each path. We thus have $Q' = Q$, $q'_0 = q_0$ and

$$\delta' = \{\, q_0 \xrightarrow{T_d(A_1)} q_1 \mid q_0 \longleftrightarrow A A_1 q_1 \,\} \cup \{\, q \xrightarrow{B} q' \mid q \longleftrightarrow A B q' \wedge q \neq q_0 \,\}.$$

If $\mathrm{l}(d) = n$, we distinguish three cases according to the nature of d:

1. If $d = (a, push_1^w)$, then $Q' = Q \cup \{q_0'\}$ and $\delta' = \delta \cup \{\, q_0' \xrightarrow{a} q_1 \mid q_0 \longleftrightarrow Awq_1 \,\}$.
2. If $d = (a, push_n)$ and $n > 1$, then $Q' = Q \cup \{q_0'\}$ and $\delta' = \delta \cup \{\, q_0' \xrightarrow{B} q_2 \mid \exists q_1,\ q_0 \longleftrightarrow AA_1q_1 \longleftrightarrow AA_2q_2 \,\}$ where $B = A_1 \times A_2 \times A_a^{(n-1)}$.
3. If $d = (a, pop_n)$, then $Q' = Q \cup \{q_0'\}$ and $\delta' = \delta \cup \{\, q_0' \xrightarrow{A_a^{(n-1)}} q_0 \,\}$.

It is not difficult to prove that $L(A'_d) = pre_d(L(A))$. Hence, if γ is the set of transitions of H, then we have $pre(S) = pre(L(A)) = \bigcup_{d\in\gamma} L(A'_d)$.

This technique can be extended to compute the set $pre^*(S)$ of all predecessors of a regular set of stacks S.

Theorem 5.4. *Given a n-HCFP H and a regular set of n-stacks S, the set $pre^*(S)$ is regular and effectively computable.*

To compute $pre^*(S)$, we have to deal with the problem of termination. A simple iteration of our previous construction will in general not terminate, as each step would add control states to the automaton. As a matter of fact, even the sequence $(pre^i(S))_{i\geq 0}$, defined as $pre^0(S) = S$ and for all $n \geq 1$ $pre^n(S) = pre^{n-1}(S) \cup pre(pre^{n-1}(S))$, does not reach a fix-point in general. For instance, if $d = (a, pop_1)$, then for all n, $pre^n([a]) = \{\, [a^i] \mid i \leq n \,\} \neq pre^{n+1}([a])$.

To build $pre^*(S)$ for some regular S, we modify the previous construction in order to keep constant the number of states in the nested automaton we manipulate. The idea, instead of creating new control states, is to add edges to the automaton until saturation, eventually creating loops to represent at once multiple applications of a HCFP transition. Then, we prove that this new algorithm terminates and is correct.

Let us first define operation T_d for any n-HCFP transition d (see Figure 5.1 for an illustration). Let $A = (Q, \Gamma, \delta, q_0, q_f)$ and $A' = (Q, \Gamma, \delta', q_0, q_f)$ be nested n-stack automata over $\Gamma' = \Gamma \cup \{\, [,] \,\}$, and d a n-HCFP transition. We define $A' = T_d(A)$ as follows.
If the level of d is less than n, then we simply propagate the transformation to the first level $n-1$ automaton encountered along each path:

$$\delta' = \{\, q_0 \xrightarrow{T_d(A_1)} q_1 \mid q_0 \longleftrightarrow AA_1q_1 \,\} \cup \{\, q \xrightarrow{B} q' \mid q \longleftrightarrow ABq' \wedge q \neq q_0 \,\}.$$

If $\mathrm{l}(d) = n$ then as previously we distinguish three cases according to d:

1. If $n = 1$ and $d = (a, push_1^w)$, then $\delta' = \delta \cup \{\, q_0 \xrightarrow{a} q_1 \mid q_0 \longleftrightarrow Awq_1 \,\}$.
2. If $d = (a, push_n)$ for some $n > 1$, then $\delta' = \delta \cup \{\, q_0 \xrightarrow{B} q_2 \mid \exists q_1,\ q_0 \longleftrightarrow AA_1q_1 \longleftrightarrow AA_2q_2 \,\}$ where $B = A_1 \times A_2 \times A_a^{(n-1)}$.

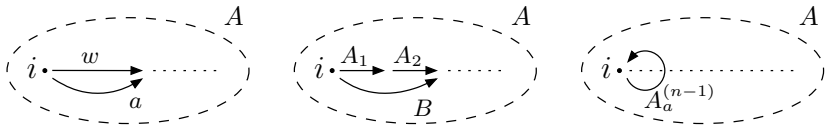


Figure 5.1: transformation $T_d(A)$ for $d = (a, push_1^w)$, $(a, push_k)$ and (a, pop_k).

---

3. If $d = (a, pop_n)$, then $\delta' = \delta \cup \{\, q_0 \xrightarrow{A_a^{(n-1)}} q_0 \,\}$

Suppose $H = (\Gamma, \delta)$ with $\delta = \{\, d_0, \ldots, d_{l-1} \,\}$. Given an automaton A such that $S = L(A)$, consider the sequence $(A_i)_{i \geq 0}$ defined as $A_0 = A$ and for all $i \geq 0$ and $j = i \mod l$, $A_{i+1} = T_{d_j}(A_i)$. In order to obtain the result, we have to prove that this sequence always reaches a fix-point (Lemma 5.5) and this fix-point is an automaton actually recognizing $pre^*(S)$ (Lemmas 5.8 and 5.9).

5.3.2.1 Termination and Complexity

Lemma 5.5 (Termination)**.** *For all nested n-stack automaton A and n-HCFP $H = (\Gamma, \delta)$, the sequence $(A_i)_{i \geq 0}$ defined with respect to A eventually stabilizes: $\exists k \geq 0,\ \forall k' \in \delta,\ A_{k'} = A_k$, which implies $L(A_k) = \bigcup_{i \geq 0} L(A_i)$.*

Proof. First, notice that for all d, T_d does not change the set of control states of any automaton occurring in A, and only adds transitions. This means $(A_i)_{i \geq 0}$ is monotonous in the size of each A_i.

To establish the termination of the construction, we prove that the number of transitions which can be added to A_0 is finite. Note that by definition of T_d, the number of states of each A_i is constant. Moreover, each new transition originates from the initial state of the automaton it is added to. Hence, the total number of transitions which can be added to a given automaton is equal to $|V_n| \cdot |Q|$, where V_n is the level n vocabulary and Q its set of states. Since $|Q|$ does not change, we only have to prove that V_n is finite for all n. If $n = 1$, $V_1 = \Gamma$, and the property holds. Now suppose $n > 1$ and the property holds up to level $n-1$. By induction hypothesis, V_{n-1} is finite. With this set of labels, one can build a finite number N of different level $n-1$ automata which is exponential in $|V_{n-1}| \cdot K$, where K depends on the number of level $n-1$ automata in A_0 and of their sets of control states. As each transition of a level n automaton is labeled by a product of level $n-1$ automata, then $|V_n|$ is itself exponential in N, and thus doubly exponential in $|V_{n-1}|$. Remark that, as a consequence, the number of steps of the construction is non-elementary in n. □

5.3.2.2 Soundness

The following elementary lemma expresses the simple fact that if some transition (a, pop_k) can be applied on a certain stack, then it must also be applicable to any stack with the same top-most level $k-1$ stack.

Lemma 5.6. *For all HCFP H and constant*[3] *stack expression s,*

$$\exists t,\ s \cdot t \xrightarrow{*} t \implies \forall t',\ s \cdot t' \xrightarrow{*} t'.$$

Proof. The proof is a simple induction on the size of expression s. □

Before proving the soundness of the construction of Proposition 5.4, we need a technical lemma expressing the fact that all cycles on the initial state of a nested automaton during the computation of (A_i) correspond to possible runs of the context-free process we consider.

Lemma 5.7. *For all $i \geq 0$ and nested k-stack automaton $B = (Q, \Gamma, \delta, q_0, q_f)$ occurring in A_i, whenever there exist a state $q_1 \neq q_0$, path labels w_1 and w_2, a transition label C and a path $q_0 \xrightarrow{w_1} q_0 \xrightarrow{C} q_1 \xrightarrow{w_2} q_f$ in B, then for all run*

$$A_i \overset{*}{\longmapsto} B \cdot r \overset{*}{\longmapsto} [\, w_1\, C\, w_2\,] \cdot r \overset{*}{\longmapsto} t \cdot s$$

where r is any stack expression, $w_1 \overset{*}{\longmapsto} t$ and $[\, C\, w_2\,] \cdot r \overset{*}{\longmapsto} s$, we necessarily have $s \in pre^*_H(t \cdot s)$ and

$$A_i \overset{*}{\longmapsto} B \cdot r \overset{*}{\longmapsto} [\, C\, w_2\,] \cdot r \overset{*}{\longmapsto} s.$$

Proof. Let us reason by induction on i. Assume for simplicity that no transition leads to the initial state in any automaton occurring in A. If $i = 0$, then $w_1 = \varepsilon$ and the property is trivial. Now suppose the property is true up to some rank $i \geq 0$. Call d the level k operation such that $A_{i+1} = T_d(A_i)$. Consider the following run ρ of A_{i+1}:

$$A_{i+1} \overset{*}{\longmapsto} B \cdot r \overset{*}{\longmapsto} [\, w_1\, C\, w_2\,] \cdot r \overset{*}{\longmapsto} t \cdot s \text{ with } w_1 \overset{*}{\longmapsto} t.$$

As w_1 labels a loop on the initial state of B, another possible run of A_{i+1} is:

$$A_{i+1} \overset{*}{\longmapsto} B \cdot r \overset{*}{\longmapsto} [\, C\, w_2\,] \cdot r \overset{*}{\longmapsto} s.$$

We only need to show that $t \cdot s \xrightarrow{*} s$ to conclude the proof. To do this, we will reason by induction on the number m of new level k transitions of A_{i+1} (i.e. transitions of A_{i+1} not in A_i) used in the w_1 cycle on q_0.

[3] We say a stack expression is constant when it contains no automaton.

$m = 0$: As w_1 contains no new transition, it also labels a cycle in A_i. Now, either transition C belongs to A_i or not. In the positive case, ρ is a path in A_i, hence the property is true by induction on i. In the case where C is a new transition, by definition of A_{i+1}, A_i admits the following run:

$$A_i \overset{*}{\longmapsto} B \cdot r \overset{*}{\longmapsto} [\, w_1\, u\, w_2 \,] \cdot r \overset{*}{\longmapsto} t \cdot s' \text{ with } w_1 \overset{*}{\longmapsto} t \text{ and } [uw_2] \cdot r \overset{*}{\longmapsto} s',$$

where u is equal to ε, $C_1\, C_2$ or v when d is (a, pop_k), $(a, push_k)$ or $(a, push_1^v)$ respectively. By induction on i, this run verifies the property, hence we have

$$A_i \overset{*}{\longmapsto} B \cdot r \overset{*}{\longmapsto} [\, u\, w_2 \,] \cdot r \overset{*}{\longmapsto} s' \text{ with } t \cdot s' \overset{*}{\longrightarrow} s'.$$

By Lemma 5.6, this implies that $\forall s'',\ t \cdot s'' \overset{*}{\longrightarrow} s''$, and in particular $t \cdot s \underset{*}{\overset{H}{\hookrightarrow}} s$.

$m \Rightarrow m + 1$: Suppose the w_1 cycle in B contains $m+1$ new transitions. Let $q_0 \longleftrightarrow D q_0$ be one of these new transitions, we have $w_1 = w_1' C w_2'$. Hence B has a path

$$q_0 \longleftrightarrow B w_1' q_0 \longleftrightarrow B D q_0 \longleftrightarrow B w_1'' q_0 \longleftrightarrow B C q_1 \longleftrightarrow B w_2 q_f$$

which begins with a cycle on q_0 labeled by w_1', containing m or less new transitions of A_{i+1}. Suppose $t = t_1 \cdot t_2$ and $w_1' \overset{*}{\longmapsto} t_1$, by induction hypothesis on m we have:

$$A_{i+1} \overset{*}{\longmapsto} B \cdot r \overset{*}{\longmapsto} [\, D\, w_1''\, C\, w_2 \,] \cdot r \overset{*}{\longmapsto} t_2 \cdot s$$

and $s \in pre_H^*(t_1 \cdot s)$. We now have to examine the way transition D is created in A_{i+1}, which depends on the type of d. As previously, by definition of A_{i+1} there must be a run of the form

$$A_i \overset{*}{\longmapsto} B \cdot r \overset{*}{\longmapsto} [\, u\, w_1''\, C\, w_2 \,] \cdot r \overset{*}{\longmapsto} t_3 \cdot s,$$

where u is equal to ε, $D_1\, D_2$ or v when d is (a, pop_k), $(a, push_k^v)$ or $(a, push_1^v)$ respectively. It is easy to show that t_3 can be chosen to be $d(t_2)$. This run uses a path in B starting with a cycle on q_0 labeled by $u\, w_1''$ which contains m or less new level k transitions:

$$q_0 \longleftrightarrow B u q_0 \longleftrightarrow B w_1'' q_0 \longleftrightarrow B C q_1 \longleftrightarrow B w_2 q_f.$$

Using the induction hypothesis on m, we can now conclude that:

$$A_{i+1} \overset{*}{\longmapsto} [\, C\, w_2 \,] \cdot r \overset{*}{\longmapsto} s \text{ and } t_3 \cdot s \in pre_H^*(s).$$

We have $t \cdot s \underset{*}{\overset{H}{\hookrightarrow}} t_2 \cdot s \longrightarrow t_3 \cdot s \underset{*}{\overset{H}{\hookrightarrow}} s$, hence $t \cdot s \underset{*}{\overset{H}{\hookrightarrow}} s$, which concludes the proof.

□

Lemma 5.8 (Soundness)**.** $\bigcup_{i\geq 0} L(A_i) \subseteq pre^*_H(S)$.

Proof sketch. We prove by induction on i the equivalent result that $\forall i,\ L(A_i) \subseteq pre^*_H(S)$. The base case is trivial since by definition $A_0 = A$ and $L(A) = S \subseteq pre^*_H(S)$. For the inductive step, we consider a stack s accepted by a run in A_{i+1} and reason by induction on the number m of new level k transitions used in this run, where k is the level of the operation d such that $A_{i+1} = T_d(A_i)$. The idea is to decompose each run containing m new transitions into a first part with less than m new transitions, one new transition, and a second part also containing less than m new transitions. Then, by induction hypothesis on m and i, one can re-compose a path in A_i recognizing some stack s' such that $s' \in pre^*_H(S)$ and $s \in pre^*_H(s')$.

Assume for simplicity that no transition of an automaton occurring in A leads to its initial state. We reason by induction on i. The base case is trivial since $A_0 = A$ and $L(A) \subseteq pre^*_H(L(A))$. Now consider a stack s in $L(A_{i+1})$. If s is accepted by A_{i+1} using no new transition, then it is accepted by A_i. Hence by induction hypothesis it belongs to $pre^*_H(S)$. Otherwise, the accepting run must be of the form

$$A_{i+1} \stackrel{*}{\longmapsto} B \cdot r \stackrel{*}{\longmapsto} [\, w_1\, C\, w_2\,] \cdot r \stackrel{*}{\longmapsto} s,$$

where the path in B which generates $w_1\, C\, w_2$ is of the form

$$q_0 \longleftrightarrow Bw_1q_0 \longleftrightarrow BCq_1 \longleftrightarrow Bw_2q_f,$$

with $q_1 \neq q_0$. By Lemma 5.7 there exist t, s_1 such that $s = t \cdot s_1$, $t \cdot s_1 \stackrel{*}{\longrightarrow} s_1$ and

$$A_{i+1} \stackrel{*}{\longmapsto} B \cdot r \stackrel{*}{\longmapsto} [\, C\, w_2\,] \cdot r \stackrel{*}{\longmapsto} s_1.$$

Note that by definition of T_d, all new transitions start from the initial states of automata in A_{i+1}. Hence, if the transition labeled by C in the previous run is not new, then the whole run exists in A_i. By induction hypothesis on i, there exists $s_2 \in S$ such that $s_1 \stackrel{*}{\longrightarrow} s_2$, hence by transitivity $s \stackrel{*}{\longrightarrow} s_2$.
If the transition labeled by C is new, then since $q_1 \neq q_0$ and by definition of T_d, d must be of the form $(a, push_k)$ or $(a, push_1^v)$. Then by construction of A_{i+1} there is a run

$$A_{i+1} \stackrel{*}{\longmapsto} B \cdot r \stackrel{*}{\longmapsto} [\, u\, w_2\,] \cdot r \stackrel{*}{\longmapsto} s_2,$$

where u is either $C_1\, C_2$ if $k > 1$ or v is $k = 1$, and s_2 can be chosen as $d(s_1)$. Now by induction hypothesis on i, there exists $s_3 \in S$ such that $s_2 \stackrel{*}{\longrightarrow} s_3$, hence by transitivity $s \stackrel{*}{\longrightarrow} s_3$.

□

5.3.2.3 Completeness

Lemma 5.9 (Completeness)**.** $pre^*_H(S) \subseteq \bigcup_{i\geq 0} L(A_i)$.

Proof sketch. We prove the sufficient property that for all nested stack automaton A and HCFP transition d, $pre_d(L(A)) \subseteq L(T_d(A))$. Consider automata A and A' such that $A' = T_d(A)$. Consider a stack $s \in L(A)$, and let s' be any stack such that $s' \in pre_{d_j}(s)$. There is a run ρ of A recognizing s as follows:

$$A \overset{*}{\longmapsto} B \cdot r \longmapsto [\, C_1 \dots C_l] \cdot r \overset{*}{\longmapsto} s.$$

Depending on d, we have to consider three cases:

1. If $d_j = (a, pop_k)$, then $s' = t \cdot s$ where t is any stack of level $k-1$ such that $top(t) = a$, and by definition of T_d the following run exists:
$$A' \overset{*}{\longmapsto} [\, A_a^{(k-1)} C_1 \dots C_l\,] \cdot r \overset{*}{\longmapsto} s'.$$

2. If $d_j = (a, push_k)$, $k > 0$, then $s = tt \cdot r$ and $s' = t \cdot r$ where $top(t) = a$ and t is in both $L(C_1)$ and $L(C_2)$. Hence t is also accepted by the level $k-1$ automaton $C_1 \times C_2 \times A_a^{k-1}$. Thus, by definition of T_d the following run exists:
$$A' \overset{*}{\longmapsto} [\, C_1 \times C_2 \times A_a^{k-1}\, C_3 \dots C_l\,] \cdot r \overset{*}{\longmapsto} s'.$$

3. If $d_j = (a, push_1^w)$, then $s = w \cdot r$ and $s' = a \cdot r$. This means $C_1 \dots C_l$ are level 0 automata (i.e. letters), and $C_1 \dots C_{|w|} = w$. By definition of T_d the following run exists:
$$A' \overset{*}{\longmapsto} [\, a\, C_{|w|+1} \dots C_l\,] \cdot r \overset{*}{\longmapsto} s'.$$

This establishes the fact that T_d adds to the language L of its argument *at least* the set of direct predecessors of stacks of L by operation d. □

As a direct consequence of Proposition 5.3 and Theorem 5.4, we obtain a symbolic model checking algorithm for the logic $\mathsf{E}(\mathsf{F}, \mathsf{X})$ with regular stack languages as atomic predicates, i.e. the fragment of the temporal logic CTL for the modal operators EF (there exists a path where eventually a property holds) and EX (there exist an immediate successor satisfying a property).

Theorem 5.10. *For every HCFP H and formula φ of $\mathsf{E}(\mathsf{F}, \mathsf{X})$, the set of configurations (stacks) satisfying φ is regular and effectively computable.*

5.4 Constraining Reachability

In this section we address the more general problem of computing a finite automaton recognizing $pre^*_H[C](S)$ for any HCFP H and pair of regular stack languages C and S. We provide an extension of the construction of Proposition 5.4 allowing us to ensure that we only consider runs of H whose configurations all belong to C. Again, from a given automaton A, we construct a sequence of automata whose limit recognizes exactly $pre^*_H[C](L(A))$. The main (and only) difference with the previous case is that we need to compute language intersections at each iteration without invalidating our termination arguments (i.e. without adding any new states to the original automaton). For this reason, we use a class of *alternating* automata, which we call *constrained* nested automata.

5.4.1 Constrained Nested Automata

Définition 5.6 (Constrained nested automata)**.** *Let B be a non-nested m-stack automaton*[4] *(with $m \geq n$). A level n B-constrained nested automaton A is a nested automaton $(Q_A, \Gamma, \delta_A, i_A, f_A)$ with special transitions of the form $p \longleftrightarrow AC(q,r)$ where $p, q \in Q_A$, r is a control state of B and C is a level $n-1$ B-constrained nested automaton.*

The language of a constrained nested automaton is defined *via* a simple adaptation of the construction of Proposition 5.1. Consider a nested automaton $A = (Q, \Gamma, \delta, i, f)$ of level n constrained with respect to a level 1 n-stack automaton $B = (Q_B, \Gamma', \delta_B, i_B, f_B)$ [5]. First, consider the (unconstrained) nested automaton $A' = (Q, \Gamma, \delta', i, f)$, where $\delta' = \{\, p \xrightarrow{C} q \mid p \longleftrightarrow AC(q,r) \,\}$. Second, build according to the construction of Proposition 5.1 a level 1 automaton $\flat A' = (\flat Q', \Gamma', \flat\delta', i', f')$ with the same accepted language as A'. By adding to $\flat A'$ the control states of B and integrating into it the set of constrained transitions of A, one gets an alternating stack automaton $\flat A = (\flat Q, \Gamma', \flat\delta, (i' \wedge i_B), f')$, where $\flat Q = \flat Q' \cup Q_B$. By construction, control states in $\flat Q'$ are of the form $q_n \dots q_k$ where $k \in [1, n]$ and each q_i is a control state of a level k automaton occurring in A'. We define $\flat\delta$ as the union of δ_B and the set of all $s \xrightarrow{x} t$ such that:

1. $\bar{p}pr \xrightarrow{x} \bar{p}q \in \flat\delta'$, $s = \bar{p}pr$, $t = (\bar{p}q \wedge u)$, $X =]$ and $p \longleftrightarrow CD(q,u)$ where C occurs in A and r is a control state of D,
2. $\bar{p}p \xrightarrow{x} \bar{p}q' \in \flat\delta'$, $s = \bar{p}p$, $t = (\bar{p}q \wedge u)$, $X = a$, and $p \longleftrightarrow Ca(q,u)$ where C is a level 1 automaton occurring in A,
3. $s \xrightarrow{x} t \in \flat\delta'$ in all other cases.

[4] i.e. a standard, level 1 finite state automaton.

[5] note that the levels of A and B have to be the same for $L(A)$ to be defined.

We now define the language accepted by A as the language accepted by the alternating automaton $\flat A$ we just defined, according to the usual notion of acceptance for alternating automata: $L(A) = L(\flat A)$ (please note that the initial state of $\flat A$ is $i' \wedge i_B$).

The intuitive idea is quite simple. Suppose A is a B-constrained nested n-stack automaton, and B also recognizes n-stacks. First, we require all the words accepted by A to be also accepted by B: $L(A) \subseteq L(B)$. Then, in any run of A where a transition of the form $p \xrightarrow{D} (q, r)$ occurs, the remaining part of the input word should be accepted both by A when resuming from state q and by B when starting from state r. Of course, when expanding D into a word of its language, it may require additional checks in B. As a matter of fact, constrained nested automata can be transformed into equivalent level 1 alternating automata. As such, the languages they accept are all regular.

Proposition 5.11. *Constrained nested automata accept regular languages.*

5.4.2 Constrained Reachability Analysis

The construction we want to provide needs to refer to whole sets of paths in a level 1 stack automaton recognizing the constraint language. To do this, we need to introduce a couple of additional definitions and notations.

Définition 5.7. *Let A be a finite stack automaton over $\Gamma' = \Gamma \cup \{ [\, ,] \}$. A state p of A is of level 0 if it has no successor by $[$ and no predecessor by $]$. It is of level k if all its successors by $[$ and predecessors by $]$ are of level $k-1$. The level of p is written $\mathrm{l}(p)$.*

We can show that any automaton recognizing only n-stacks is equivalent to an automaton whose control states all have a well-defined level. A notion of level can also be defined for paths. A *level n path* in a stack automaton is a path $p_1 \dots p_k$ with $\mathrm{l}(p_1) = \mathrm{l}(p_k) = n$ and $\forall i \in [2, k-1],\ \mathrm{l}(p_i) < n$. All such paths are labeled by n-stacks. Now, to concisely refer to the whole set of level n paths between two level n control states, we introduce the following notation. Let

$$Q = \{\, q \in Q_A \mid \mathrm{l}(q) < n \wedge p_1 \longleftrightarrow A{+}q \longleftrightarrow A{+}p_2 \,\}$$

be the set of all states of A occurring on a level n path between p_1 and p_2. If Q is not empty, we write $p_1 \overset{B}{\underset{A}{\rightsquigarrow}} p_2$, where B is defined as:

$$B = \big(\, Q_B = Q \cup \{p_1, p_2\},\ \Gamma',\ \delta_B = \delta_A \cap (Q_B \times \Gamma' \times Q_B),\ p_1,\ p_2 \,\big).$$

Thanks to these few notions, we can state our result:

Theorem 5.12. *Given a n-HCFP H and regular sets of n-stacks S and C, the set $pre^*_H[C](S)$ is regular and effectively computable.*

To address this problem, we propose a modified version of the construction of the previous section, which uses constrained nested automata. Let $d = (a, o)$ be a HCFP transition rule, $A = (Q_A, \Gamma, \delta, i, f)$ and $A' = (Q_A, \Gamma, \delta', i, f)$ two nested k-stack automata constrained by a level 1 n-stack automaton $B = (Q_B, \Gamma', \delta_B, i_B, f_B)$ accepting C (with $n \geq k$). We define a transformation $T^B_{d_j}(A)$, which is very similar to T_{d_j}, except that we need to add alternating transitions to ensure that no new stack is accepted by A' unless it is the transformation of a stack previously accepted by B (Cf. Figure 5.2). If $\mathrm{l}(d) < k$, we propagate the transformation to the first level $k-1$ automaton along each path:

$$\delta' = \{\, i \xrightarrow{T^B_d(C)} (p,q) \mid i \longleftarrow AC(p,q) \,\} \cup \{\, p \xrightarrow{C} (p',q') \in \delta \mid p \neq i \,\}.$$

If $\mathrm{l}(d) = n$, we distinguish three cases according to the nature of d:

1. If $d = (a, push^w_1)$, then

$$\delta' = \delta \cup \Big\{\, i \xrightarrow{a} (p,q) \mid i \longleftarrow A_i w(p,q') \quad \wedge \quad \exists q_1, q \in Q_B, \\ \mathrm{l}(q_1) = \mathrm{l}(q) = 0,\ i_B \longleftarrow B[^n q_1 \longleftarrow Bwq \,\Big\}.$$

2. If $d = (a, push_k)$, then for $m = n - k + 1$ and $C = (C_1 \times C_2) \times (B_1 \times B_2) \times A^{(k-1)}_a$,

$$\delta' = \delta \cup \Big\{\, i \xrightarrow{C} (p,q) \mid i \longleftarrow A_i C_1 \longleftarrow A_i C_2(p,q') \quad \wedge \quad \exists\, q_1,\, q_2,\, q \in Q_B, \\ \mathrm{l}(q_1) = \mathrm{l}(q_2) = \mathrm{l}(q) = k-1,\ i_B \longleftarrow B[^m q_1 \overset{B_1}{\underset{B}{\rightsquigarrow}} q_2 \overset{B_2}{\underset{B}{\rightsquigarrow}} q \,\Big\}.$$

3. If $d = (a, pop_k)$, then for $m = n - k + 1$,

$$\delta' = \delta \cup \Big\{\, i \xrightarrow{A^{(k-1)}_a} (i,q) \mid \exists q \in Q_B,\ \mathrm{l}(q) = k-1,\ i_B \longleftarrow B_1[^m q \,\Big\}.$$

Suppose $H = (\Gamma, \delta)$ with $\delta = \{\, d_0, \ldots, d_{l-1} \,\}$. Given an automaton A such that $S = L(A)$, consider the sequence $(A_i)_{i \geq 0}$ defined as $A_0 = A^B$ (the B-constrained automaton with the same set of states and transitions as A, whose language is $L(A) \cap L(B)$) and for all $i \geq 0$ and $j = i \mod l$, $A_{i+1} = T^B_{d_j}(A_i)$. By definition of T^B_d, the number of states in each A_i does not vary, and since the number of control states of B is finite the same termination arguments as in Lemma 5.5 still hold. It is then quite straightforward to extend the proofs of Lemma 5.8 and Lemma 5.9 to the constrained case.

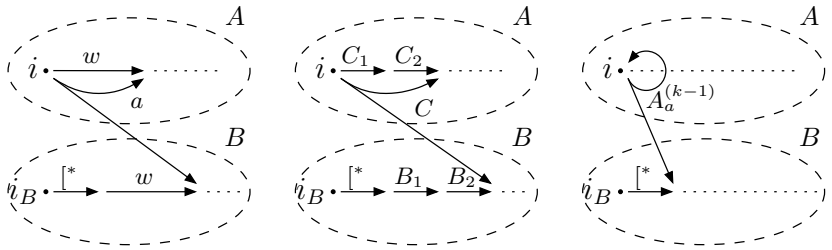


Figure 5.2: transformation $T_d^B(A)$ for $d = (a, push_1^w)$, $(a, push_k)$ and (a, pop_k).

Lemma 5.13 (Termination)**.** *For all nested n-stack automaton A, level* 1 *n-stack automaton B and n-HCFP* $H = (\Gamma, \delta)$*, the sequence* (A_i^B) *defined with respect to A and B eventually reaches* $T_H^B(A)$*:*

$$\exists k \geq 0,\ \forall d \in \delta,\ T_d^B(A_k^B) = A_k^B.$$

Proof. The algorithm for computing $T_H^B(A)$ is similar to the one for computing $T_H(A)$, except that it labels some of the transitions of each A_i^B by a state of B. As the number of such states remains unchanged throughout the whole computation, this does not add any unboundedness in the computation and the maximal number of iterations before reaching a fix-point is still finite. □

Lemma 5.14 (Soundness)**.** $\forall i,\ L(A_i^B) \subseteq pre_H^*[C](S)$.

Proof. By definition of $L(A_i^B)$, $L(A_i^B) \subseteq L(A_i)$ for all i. So, by Lemma 5.8, we already have $L(A_i^B) \subseteq pre_H^*(S)$. Let us reason by induction on i. By definition of constrained nested automata, $L(A_0^B) = L(A) \cap C$, hence $L(A_0^B) \subseteq pre_H^*[C](S)$. Now assume the property is true up to some rank i, and consider the automaton A_{i+1}^B. Note that everywhere transformation T_d^B adds a transition in A_i^B to get A_{i+1}^B, the alternating transitions induced in $\flat A_{i+1}^B$ ensure that each stack labeling a new accepting path in the automaton is a transformation of a stack labeling an accepting path in B. This way, one makes sure that no element of C in $pre_H^*(S) \setminus pre_H^*[C](S)$ is added to the language of A_{i+1}^B.

For instance, assume $d = (a, push_1^w)$ and some stack s is accepted by $\flat A_{i+1}^B$ using a a-transition newly created by T_d^B. According to the definition of T_d^B, this transition is of the form $p \xrightarrow{a} (q, r)$, where r is a control state reachable in B through a path labeled by $\lceil^n w$. Thus, if we let $w(s) = \lceil^n a\, w'$, for s to be accepted by $\flat A_{i+1}^B$, then necessarily s' must be accepted by B from state r. The same kind of reasoning holds for the other types of operations. □

Lemma 5.15 (Completeness)**.** $\forall i,\ S_i^C \subseteq L(A_i^B)$.

Proof. By definition, $L(A_0^B) = S \cap C = S_0^C$. Now suppose the property is true up to some rank i, and consider a stack $s \in S_{i+1}^C \setminus S_i^C$. Let d be the operation such that $S_{i+1}^C = S_i^C \cup (d(S_i^C) \cap C)$. By definition, there is a stack $s' \in S_i^C$ such that $s' = d(s)$, and by induction hypothesis s' is accepted by A_i^B. Moreover, since both s and s' are in C, they are accepted by B. As seen in Lemma 5.9, transformation T_d adds a new transition $p_0 \xrightarrow{C} q$ creating in particular a path labeled by s. The additional constraints T_d^B puts on this transition, and all paths in A_{i+1}^B in general, forbids any path labeled by some r using this transition to be accepted unless both r and $d(r)$ also have an accepting run in B. This is the case for s and s', hence $s \in L(A_{i+1}^B)$. □

This more general construction also allows us to extend Theorem 5.10 to the larger fragment $\mathsf{E}(\mathsf{U},\mathsf{X})$ of CTL, where formulas can now contain the modal operator EU (there exists a path along which a first property continuously holds until a second property eventually holds) instead of just EF.

Theorem 5.16. *Given a HCFP H and formula φ of $\mathsf{E}(\mathsf{U},\mathsf{X})$, the set of configurations (stacks) satisfying φ is regular and effectively computable.*

5.5 Conclusion

We have provided an automata-based symbolic technique for backward reachability analysis of higher-order context-free processes. This technique can be used to check temporal properties expressed in the logic $\mathsf{E}(\mathsf{U},\mathsf{X})$. In this respect, our results provide a first step toward developing symbolic techniques for the model-checking of higher-order context-free or pushdown processes.

Several important questions remain open and are left for future investigation. In particular, it would be interesting to extend our approach to the more general case of higher-order pushdown systems, i.e. by taking into account a set of control states. This does not seem to be technically trivial, and naive extensions of our construction lead to procedures which are not guaranteed to terminate.

Another interesting issue is to generalize our symbolic approach to more general properties than reachability and/or safety, including liveness properties. Finally, it would also be very interesting to extend our symbolic techniques in order to solve games (such as safety and parity games) and to compute representations of the sets of all winning configurations for these games.

Another approach to this problem was independently undertaken by Arnaud Carayol [Car05] using a different notion of regular sets of configurations. His results allow a full forward and backward symbolic reachability analysis of the general class of higher-order pushdown systems. However, the problem is shown

to be inherently non-elementary, which seems to forbid any practical application. It might be interesting to investigate the precise complexity gain when only considering higher-order context-free processes, and compare both approaches in that case. It still remains to determine whether any useful class of systems between pushdown and higher-order pushdown systems with a reachability analysis of elementary complexity can be found.

Appendix A

Finite Acceptors for Context-Sensitive Languages

The aim of this appendix is to present the transformations that link different acceptors of the context-sensitive languages considered in this article and to convince the reader that they are syntactical. It also aims at establishing tight complexity results for the translations between these acceptors. In order to simplify our presentation, we only consider context-sensitive languages that do not contain the empty word ε.

A.1 Unlabeled Linearly Bounded Machines

Usual definitions of linearly bounded machines have unlabeled transitions, they start their computations in a configuration where the input word is written on the tape, and they are neither able to insert not delete tape cells. Despite these differences, both definition of LBMs coincide. In the following proposition, we refer to our definition of linearly bounded machines as *labeled* LBMs.

Proposition A.1. *A language is (deterministic) context-sensitive if and only if it is accepted by a (deterministic) labeled linearly bounded Turing machine.*

Proof. Let us first consider a context-sensitive language L accepted by a terminating unlabeled LBM N (by Prop. 1.10). We sketch the construction of a labeled LBM $M = (\Gamma, \Sigma, [,], Q, q_0, F, \delta)$ accepting L.

Let q_A, q_R and q_S be three states in Q. In a configuration of the form $[wq_X]$ for $w \in \Sigma^*$ and $q_X \in \{q_A, q_R\}$, M reads an input letter $a \in \Sigma$ and goes to the configuration $[waq_s]$. Then it simulates N on wa using only ε-transitions while remembering wa. Using the same work alphabet as N, this would require the use of $2|wa|$ cells. However using an increased work alphabet, this can be done using only $|wa|$ cells. If N accepts (resp. rejects) wa, M restores wa on its tape and

enters the configuration $[waq_A]$ (resp. $[waq_R]$). By taking $F = \{q_A\}$ and $q_0 = q_A$ if ε belongs to L and $q_0 = q_R$ otherwise, it is easy to see that the set of words accepted by M from $[q_0]$ is precisely L. Note that non-deterministic behavior can only appear in M while simulating N. Hence, if N is deterministic then so is M.

Conversely, let M be a labeled LBM accepting a language $L \subset \Sigma^*$. We describe an unlabeled LBM N accepting L with two tapes: the input tape and work tape. It is well known that this model is equivalent to unlabeled LBMs with one tape (see for instance [HU79]). To each tape corresponds a set of control states: the set of work states Q_Γ contains the set Q of states of N and the set of input states Q_Σ is reduced to $\{q_i, q_A\}$ respectively the initial and accepting state.

The machine N working on word $w \in \Sigma^*$ starts with the configuration $([q_i w], [q_0])$. From a configuration $([w_1 q_i w_2], [uqv])$ with $w_1, w_2 \in \Sigma^*$, $q \in Q$ and $u, v \in \Gamma^*$, N can non-deterministicaly simulate any ε-transition of N that can be applied to the configuration represented by the work tape. The deletion rules are simulated by shifting all tape content on the right of the read head by one cell to the left. Moreover N can non-deterministically simulate any a-transition for $a \in \Sigma$ provided that the input head is on top of the symbol a in which case it is moved one cell to the right. The insertion rules are simulated by shifting the work tape content to the right of the read head by one cell on the right.

The machine N enters the accepting state q_A if the input head is on top of the right border symbol] and the work state is a final state of M. It is follows from the construction of N that w is accepted if and only if it is accepted by M. Moreover, if M is deterministic then so is N. □

A.2 One-Way Cellular Automata

In a cellular automaton, the next value of a cell depends on its current value and on the current value of its left and right neighbours. If we only allow the next value of a cell to depend on its current value and on the current value of its left neighbour, we obtain the notion of *one-way cellular automaton* which has the same expressive power (see Theorem 4.6).

Definition A.2. A *one-way cellular automaton* C is a tuple $(\Gamma, \Sigma, F, [,], \delta)$ where:

- Γ and $\Sigma \subseteq \Gamma$ are the work and input alphabet,
- $F \subseteq \Gamma$ is the accepting alphabet,
- [and] are symbols which do not belong to Γ,
- $\delta \subseteq (\{ [\} \cup \Gamma) \times \Gamma \times \Gamma) \cup (\Gamma \times \{] \} \times \{] \})$ is the transition function.

For all $u, v \in \Gamma^+$, a configuration $c' = [u]$ is a successor of a configuration $c = [v]$ if $|c| = |c'| = n$ and for all $i \in [1, n-1]$, $(c(i), c(i+1), c'(i+1)) \in \delta$.

In the following, we use the following graphical representation for a transition $(A, B, C) \in \delta$:

A	B
	C

A.3 General Expressiveness Results

We present well-known simulation results between the various acceptors defined above: linearly bounded Turing machines, cellular automata, one way cellular automata and tiling systems.

Theorem A.3. *For any language L, the following facts are equivalent:*

1. *L is a context-sensitive language,*
2. *L is recognised by a linearly bounded Turing machine,*
3. *L is recognised by a cellular automaton,*
4. *L is recognised by a one way cellular automaton,*
5. *L is recognised by a tiling system.*

Proof. We prove the following implications:

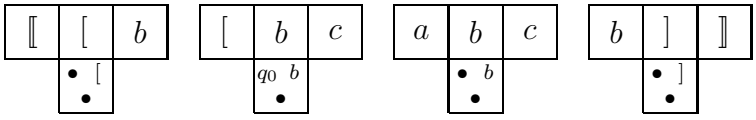
$1 \Leftrightarrow 2$: As mentioned earlier, the context-sensitive languages were originally defined as the languages generated by *growing* or *context-sensitive* grammars. The link with the languages accepted by linearly bounded Turing machine was established in [Kur64].

$2 \Rightarrow 3$: Given a linearly bounded Turing machine $M = (\Gamma, \Sigma, [,], Q, q_0, F, \delta)$, we can assume without loss of generality that M is such that from any accepting configuration there is a transition to a blocking, non-final configuration. This behaviour can be obtained by adding to Q a new state $q_\perp$ and all possible transitions from state in F to $q_\perp$. We define a cellular automaton $C = (\Gamma', \Sigma, [\![,]\!], F', \delta')$ recognising the language $[L(M)]$. Let

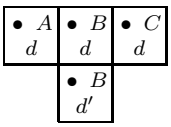
$$\Gamma' = \Big(\Sigma \ \cup \ \{[,]\}\Big) \ \cup \ \Big((Q \cup \bullet) \times \big(\Gamma \cup \{[,]\}\big) \times (\delta \cup \bullet)\Big)$$

be the work alphabet of C. Each letter in Γ' is either a letter in Σ, a border symbol [or], or a triple (c, A, d) where c is a control state of M or the special character $\bullet$, A is one of M's tape symbols, and d the last transition used or the special character $\bullet$.

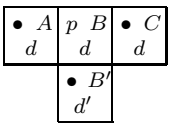
First, we specify the set of transitions δ' of C. We need transitions to encode the initial configuration of M starting from the input word:



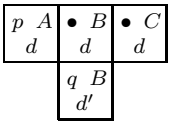
where $b \in \Sigma$ and $a, c \in \Sigma \cup \{[,]\}$. We now need to describe how the cellular automaton simulates one computation step of the machine:



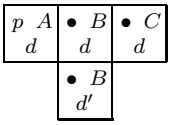
with d' any transition in δ,



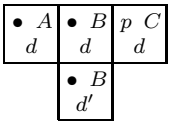
with $d' = pB \to qB'\pm \in \delta$,



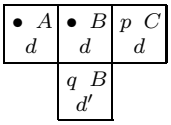
with $d' = pA \to qA'+ \in \delta$,



with $d' = pA \to qA'- \in \delta$,

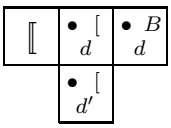


with $d' = pC \to qC'+ \in \delta$,

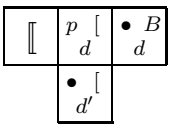


with $d' = pC \to qC'- \in \delta$,

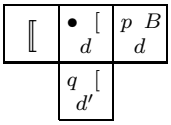
where d can be any transition or $\bullet$, $B \in \Gamma$ and $A, C \in \Gamma \cup \{[,]\}$. We also have to deal with the border of the configuration, with the following rules (only the left case is presented, the right case being symmetric):



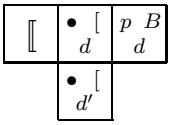
with d' any transition in δ,



with $d' = p[\to q[+ \in \delta$,



with $d' = pB \to qB'- \in \delta$,



with $d' = pB \to qB'+ \in \delta$,

where d can be any transition and $B \in \Gamma$. It is not possible to stop the computation as soon as the current configuration c contains a state in F, because this configuration has not been checked yet and might be malformed (i.e. c might not belong to $\big((Q \cup \bullet) \times (\Gamma \cup \{[,]\}) \times \{d\}\big)^*$ for some fixed

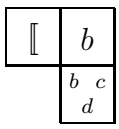
$d \in \delta$). But, we can accept if the state in the previous row was in F. Hence, the set of final states F' of C is the set of symbols in Γ' with a transition of the form $q_f A \to pB\epsilon$ where $q_f \in F$ (the existence of such a transition for each final state is guaranteed by the addition of $q_\perp$).

Let π be the morphism from $(\Gamma')^*$ to $(Q \cup \Gamma)^*$ defined by $\pi(\bullet, A, d) = A$ and $\pi(q, A, d) = qA$, it is easy to prove that there exists a run $c_0, c_1, \ldots, c_n, c_{n+1}$ of C if and only if $\pi(c_1), \ldots, \pi(c_n)$ is a run of M. Moreover, if the run $c_0, c_1 \ldots c_n, c_{n+1}$ is accepting (i.e $c_{n+1} \in [(F')^+]$), then $\pi(c_n)$ contains a final state and $\pi(c_1), \ldots, \pi(c_n)$ is also accepting. The converse is true due to the addition of the state $q_\perp$.

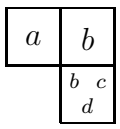
It follows that the language recognised by C is $[L(M)]$. A cellular automaton C' recognising exactly $L(M)$ is easily obtained from C by extending the work alphabet. Its time complexity is that of M plus two.

$3 \Rightarrow 4$: Let $C = (\Gamma, \Sigma, F, [,], \delta)$ be a cellular automaton. Let us construct an equivalent one-way cellular automaton $C' = (\Gamma', \Sigma, F', [\![,]\!], \delta')$, where $\Gamma' = \Sigma \cup ((\Gamma \cup \{\,]\,\}) \times \Gamma \times (\Gamma \cup \{\,\square\,\}))$. A symbol in Γ' is either an input symbol of C or a tuple (p, q, r) where p represents the current value of a cell, q a guess of the current value of its right neighbour, and r its possible next value or the symbol $\square \notin \Gamma$.

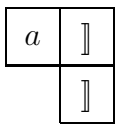
The set of transitions δ' is as follows. First, we have to guess the current value of the right neighbour of each cell, and compute a possible next value according to this guess, the value of the left neighbour, and the set of rules δ. We can also indicate that there is no next value, using the symbol $\square$:



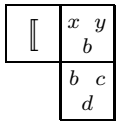
with $([, b, c, d) \in \delta$ or $d = \square$



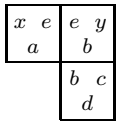
with $(a, b, c, d) \in \delta$ or $d = \square$



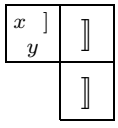
Then, we keep applying this method while additionally checking that the guess made by the left neighbour at the previous iteration was correct:



with $([, b, c, d) \in \delta$ or $d = \square$



with $(a, b, c, d) \in \delta$ or $d = \square$



where $x \in \Gamma$ and $y \in \Gamma \cup \{\,]\,\}$. The computation is successful when an accepting symbol for C appears as the *current* value of all cells (i.e. a symbol $(f, x, y) \in \Gamma'$ with $f \in F$). In other words, $F' = F \times (\Gamma \cup \{\,]\,\}) \times (\Gamma \cup \{\square\})$.

Consider an accepting run $\rho = c_1, \ldots, c_n$ of C with $c_1 = [w]$. For all $i \in [1, n]$, define $c'_i \in \Gamma'^*$ as follows. For all $i \in [1, n]$, let $c'_i(1) = [\![$ and $c'_i(|c_1|) =]\!]$. For all $i < n$ and $j \in [2, |c_1| - 1]$, let $c'_i(j) = (c_i(j), c_i(j+1), c_{i+1}(j+1))$. Finally for all $j \in [2, |c_1| - 1]$, let $c'_n(j) = (c_{n-1}(j), c_{n-1}(j+1), \Box)$. By construction of δ', the sequence $[\![w]\!], c'_1 \ldots, c'_n$ is an accepting run of C'. Thus $L(C) \subseteq L(C')$.

Conversely, let π be the projection defined as follows: $\pi([\![) = [$, $\pi(]\!]) =]$, and for all $\gamma = (p, q, r) \in \Gamma' \setminus \Gamma$, $\pi(\gamma) = p$. Then if $c_1, c_2, \ldots c_n$ is an accepting run of C' with $c_1 = [\![w]\!]$, then $\pi(c_2), \ldots, \pi(c_n)$ is an accepting run of C with $\pi(c_2) = [w]$. Thus $L(C') \subseteq L(C)$.

It follows that C' is equivalent to C. Moreover, if C accepts in time $f(n)$ then C' accepts in time $f(n) + 1$.

$4 \Rightarrow 5$: Let $C = (\Gamma, \Sigma, F, [,], \delta)$ be a one-way cellular automaton. Consider the tiling system $S = (\Gamma, \Sigma, \#, \Delta)$ such that Δ contains the following tiles:

$$\begin{array}{|c|c|}\hline \# & B \\ \hline \# & C \\ \hline \end{array} \quad \begin{array}{|c|c|}\hline \# & \# \\ \hline \# & B \\ \hline \end{array} \quad \text{for all} \quad \begin{array}{|c|c|}\hline [& B \\ \hline & C \\ \hline \end{array} \in \delta$$

$$\begin{array}{|c|c|}\hline A & B \\ \hline X & C \\ \hline \end{array} \quad \text{for all} \quad \begin{array}{|c|c|}\hline A & B \\ \hline & C \\ \hline \end{array} \in \delta$$

$$\begin{array}{|c|c|}\hline A & \# \\ \hline X & \# \\ \hline \end{array} \quad \begin{array}{|c|c|}\hline \# & \# \\ \hline A & \# \\ \hline \end{array} \quad \text{for all} \quad \begin{array}{|c|c|}\hline A &] \\ \hline &] \\ \hline \end{array} \in \delta$$

$$\begin{array}{|c|c|}\hline \# & \# \\ \hline A & B \\ \hline \end{array} \quad \text{for all} \quad A, B \in \Sigma$$

where A, B, C and X are letters in Γ. These tiles ensure that the picture describes a valid computation of C. It remains to enforce that the last row of the computation is of the form $[F^+]$. Therefore, we add the following tiles:

$$\begin{array}{|c|c|}\hline \# & f \\ \hline \# & \# \\ \hline \end{array} \quad \begin{array}{|c|c|}\hline f & f' \\ \hline \# & \# \\ \hline \end{array} \quad \begin{array}{|c|c|}\hline f' & \# \\ \hline \# & \# \\ \hline \end{array}$$

where f and f' belong to F.

$5 \Rightarrow 2$: Let $S = (\Gamma, \Sigma, \#, \Delta)$ be a tiling system. We build a linear bounded Turing machine $M = (\Gamma, \Sigma, [,], Q, p_0, F, \delta)$ recognising $L(S)$. First, let

$$Q = \{p_B^A, q_B^A \mid A, B \in \Gamma \cup \#\} \cup \{p_{0A}, p_{fA}, q_{fA} \mid A \in \Gamma \cup \#\} \cup \{p_0, p_f, q_f\}$$

with $F = \{p_f, q_f\}$. We now define δ. First, we have to provide transition rules to be able to make sure the input word is a correct first row of a picture in $P(S)$. For all $C, D \in \Sigma$, we have:

$$p_0 D \to p_{0D} D+ \quad \text{for all} \quad \begin{array}{|c|c|}\hline \# & \# \\ \hline \# & D \\ \hline \end{array} \in \Delta,$$

$$p_{0C} D \to p_{0D} D+ \quad \text{for all} \quad \begin{array}{|c|c|}\hline \# & \# \\ \hline C & D \\ \hline \end{array} \in \Delta,$$

$$p_{0C}\,] \to q^{\#}_{\#}\,] - \quad \text{for all} \quad \begin{array}{|c|c|}\hline \# & \# \\ \hline C & \# \\ \hline \end{array} \in \Delta.$$

These rules ensure that there is a partial monotonous run of M of the form $[p_0 w] \xrightarrow{*} [w_1 q^{\#}_{\#} w_2]$ where $w_1 w_2 = w$ and $w_2 \in \Sigma$ if and only if w is a possible first row of a picture in $P(S)$.

Then, we have to transform in one sweep of the read head, either from the left to the right or from the right to the left, a configuration representing a row of a picture in $P(S)$ into another configuration representing a possible adjacent row in the same picture, changing directions when the head meets a tape delimiter. For all $A, B, C, D \in \Gamma$, we thus have:

$$\left.\begin{array}{l} p^{\#}_{\#} B \to p^{B}_{D} D+ \\ q^{B}_{D}\,[\to p^{\#}_{\#}\,[+ \end{array}\right\} \quad \text{for all} \quad \begin{array}{|c|c|}\hline \# & B \\ \hline \# & D \\ \hline \end{array} \in \Delta,$$

$$\left.\begin{array}{l} p^{A}_{C} B \to p^{B}_{D} D+ \\ q^{B}_{D} A \to q^{A}_{C} C- \end{array}\right\} \quad \text{for all} \quad \begin{array}{|c|c|}\hline A & B \\ \hline C & D \\ \hline \end{array} \in \Delta,$$

$$\left.\begin{array}{l} p^{A}_{C}\,] \to q^{\#}_{\#}\,]- \\ q^{\#}_{\#} A \to q^{A}_{C} C- \end{array}\right\} \quad \text{for all} \quad \begin{array}{|c|c|}\hline A & \# \\ \hline C & \# \\ \hline \end{array} \in \Delta.$$

These rules ensure that there is a partial monotonous run $[p^{\#}_{\#} u] \longrightarrow [v_1 q^{\#}_{\#} v_2]$ of M if and only if u and $v_1 v_2$ are a pair of adjacent rows in at least one picture in $P(S)$. Conversely, this is also true of $u_1 u_2$ and v for any partial monotonous run $[u_1 q^{\#}_{\#} u_2] \longrightarrow [p^{\#}_{\#} v]$. It means that between two reversals, M replaces the current tape content seen as a picture row by a possible adjacent row according to the tiles in Δ.

Finally, we have to be able to recognise a possible last row of a picture in $P(S)$, and reach an accepting state of the machine. Again, this might be

done in both directions, according to the number of rows recognised so far:

$$\left.\begin{array}{r} p^{\#}_{\#}B \to p_{f_B}B+ \\ q_{f_B}[\to p_f[+ \end{array}\right\} \quad \text{for all} \quad \begin{array}{|c|c|}\hline \# & B \\ \hline \# & \# \\ \hline \end{array} \in \Delta,$$

$$\left.\begin{array}{r} p_{f_A}B \to p_{f_B}B+ \\ q_{f_B}A \to q_{f_A}A- \end{array}\right\} \quad \text{for all} \quad \begin{array}{|c|c|}\hline A & B \\ \hline \# & \# \\ \hline \end{array} \in \Delta,$$

$$\left.\begin{array}{r} p_{f_A}] \to q_f]- \\ q^{\#}_{\#}A \to q_{f_A}A- \end{array}\right\} \quad \text{for all} \quad \begin{array}{|c|c|}\hline A & \# \\ \hline \# & \# \\ \hline \end{array} \in \Delta.$$

These last rules ensure that there is a partial monotonous run in M from $[p^{\#}_{\#}u]$ to $[u_1q_fu_2]$ or from $[u_1q^{\#}_{\#}u_2]$ to $[p_fu]$ if and only if $u = u_1u_2$ is a possible bottom row of a picture in $P(S)$. Putting together all these observations, we conclude that $L(M) = L(S)$. Furthermore, if a word w is the frontier of a picture of height m in $P(S)$, then it is accepted by M in m reversals.

A.4 Tighter complexity bounds

The previous proofs are syntactical, and aim to express the important similarity between all four families of acceptors. However, to obtain tighter simulation results, one needs slightly more involved constructions when translating a Turing machine into a cellular automaton.

Proposition A.4. *The following simulations link linearly bounded Turing machines, cellular automata, one way cellular automata and tiling systems.*

1. *A linearly bounded Turing machine M working in $f(n)$ reversals can be simulated by a cellular automaton C working in time $f(n) + 2$.*
2. *A cellular automaton C working in time $f(n)$ can be simulated by a one-way cellular automaton C' working in time $f(n) + 1$.*
3. *A one-way cellular automaton C working in time $f(n)$ can be simulated by a tiling system S working in height $f(n)$.*
4. *A tiling system working in height $f(n)$ can be simulated by a linearly bounded Turing machine M working in $f(n)$ reversals.*

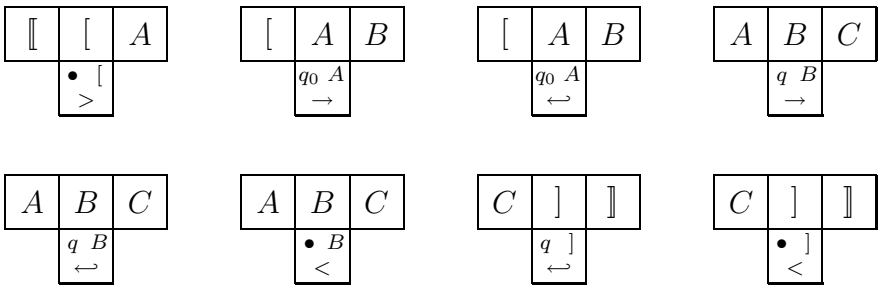
Proof.

1. Let $M = (\Gamma, \Sigma, [,], Q, q_0, F, \delta)$ be a linearly bounded Turing machine working in $f(n)$ reversals. We define a cellular automaton $C = (\Gamma', \Sigma, F', [\![,]\!], \delta')$ recognising the language $[L(M)]$ in time $f(n) + 2$. The idea of the construction is, for each possible run, to encode in a single step of the cellular

automaton all the computation steps of M between two head reversals. Hence, the length of a run of C simulating a run ρ of M should be equal to the number of head reversals in ρ, plus two for the initialisation and the final steps.

To ensure consistency, we have to check that adjacent cells agree on the direction in which the read head progresses, that the control states reached by M in each cell (or absence thereof) obey the transition rules of the Turing machine, and that no control state appears in cells which are not between the previous and next reversal. Additionally, we have to check that the symbols appearing in each cell are consistent with the symbol and control state appearing in the same cell in the previous configuration. To achieve this, it is sufficient to define the alphabet Γ' used by C as follows. Let $\Gamma' = \Sigma \cup (Q \cup \{\bullet\}) \times \Gamma \times \{\hookleftarrow, \hookrightarrow, \leftarrow, \rightarrow, <, >\}$. We use the arrow symbols $\hookleftarrow$ and $\hookrightarrow$ to denote a head reversal, $\leftarrow$ and $\rightarrow$ for a monotonous head move, and $<$ and $>$ to denote cells which are to the right (resp. the left) of the active region of the tape. Each cell also contains a control state of M, or $\bullet$ when the cell is not reached by the head at this time.

Now, we can describe the set of rules δ'. We do not give the complete set of rules but rather illustrate the construction on a few representative cases. First, we have to guess the behaviour of the Turing machine between the beginning of the run and the first reversal. Rules achieving this include:



where A, B and C are symbols in Γ and q a control state in Q. Now the first monotonous sequence of moves has been guessed, one needs to check its validity and describe the rest of the run. As there is a considerable number of rules to enumerate, we only state some of the consistency conditions which our rules must observe. A rule (W, X, Y, Z) is in δ' if and only if it satisfies *all* the following constraints (and their duals):

(a) If W (resp. X) is of the form $(\bullet, A, >)$ then X (resp. Y) must be of the form $(\bullet, B, >)$, $(p, B, \rightarrow)$ or $(p, B, \hookleftarrow)$.
(b) If W (resp. X) is of the form $(p, A, \rightarrow)$ then X (resp. Y) must be of the form $(q, B, \rightarrow)$ or $(q, B, \hookleftarrow)$ with $pA \rightarrow qA'+ \in \delta$.

(c) If W (resp. X) is of the form $(p, A, \hookleftarrow)$ then X (resp. Y) must be of the form $(\bullet, B, <)$.
(d) If W (resp. X) is of the form $(\bullet, A, <)$ then X (resp. Y) must be of the form $(\bullet, B, <)$.
(e) If X is of the form $(p, A, \rightarrow)$ then Z must be of the form $(\bullet, A', >)$ or $(p', A', \leftarrow)$ or $(p', A', \hookrightarrow)$ with $pA \rightarrow qA' + \in \delta$.
(f) If Y is of the form $(p, A, \hookleftarrow)$ then Z must be of the form $(q, B, \rightarrow)$ or $(q, B, \hookrightarrow)$ with $pA \rightarrow qB - \in \delta$.

As previously said, rules should also satisfy all dual constraints (i.e. rules constraining X with respect to Y, W with respect to X and Z with respect to W when dual arrows appear). There are also a few simple constraints concerning the configuration delimiters $[\![$ and $]\!]$. Finally, we must provide a last set of rules to be able to check that the previous configuration is correct and that it contains a final state. This is not detailed here but is a simple exercise.

2. The construction given in the proof of Theorem A.3 translates a cellular automaton working in time $f(n)$ in a one-way cellular automaton working in time $f(n) + 1$.
3. The construction given in the proof of Theorem A.3 translates a one-way cellular automaton working in time $f(n)$ in a tiling system of height $f(n)+1$.
4. The construction given in the proof of Theorem A.3 translates a tiling system of height $f(n)$ in a linearly bounded Turing machine working in $f(n)$ reversals.

A.5 Deterministic case

Finally, we present a few less standard results analogous to the previous simulations in the case of *deterministic* context-sensitive languages, which were originally characterised as the class of languages accepted by deterministic linearly bounded machines. Relevant notions of determinism can be defined for all the other acceptors, in a sense which allows them to accept exactly the class of deterministic context-sensitive languages.

Theorem A.5. *For any language L, the following facts are equivalent:*

1. *L is a deterministic context-sensitive language,*
2. *L is recognised by a deterministic linearly bounded Turing machine,*
3. *L is recognised by a deterministic cellular automaton,*
4. *L is recognised by a deterministic tiling system.*

Proof. We prove the following implications:

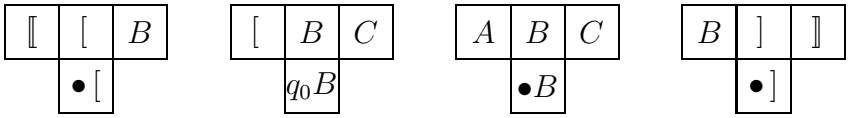
$1 \Leftrightarrow 2$: As mentioned earlier, the deterministic context-sensitive languages were originally defined as the languages accepted by the family of deterministic linearly bounded Turing machines.

$2 \Rightarrow 3$: Let $M = (\Gamma, \Sigma, [,], Q, q_0, F, \delta)$ be a deterministic linearly bounded machine. We can assume with out lost of generality that M is such that all accepting configurations are blocking, i.e. no transition is possible when the current control state is in F. We define a deterministic cellular automaton $C = (\Gamma', \Sigma, \{\bot\}, [\![,]\!], \delta')$ recognising the language $[L(M)]$. Let

$$\Gamma' = \Big(\Sigma \ \cup \ \{[,]\}\Big) \ \cup \ \Big((Q \cup \bullet) \times \big(\Gamma \cup \{[,]\}\big)\Big)$$

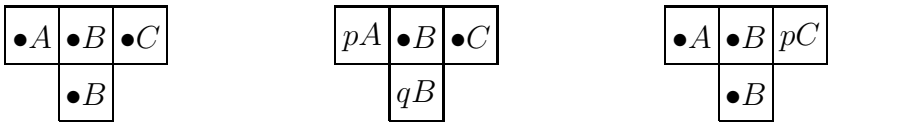
be the work alphabet of C. Each letter in Γ' is either a letter in Σ, a border symbol, or a pair (c, A) where c is a control state of M or the special character $\bullet$, and A is one of M's tape symbols. The set of transitions δ' of C is defined very similarly to the general case.

First, we need transitions to encode the initial configuration of M starting from the input word:

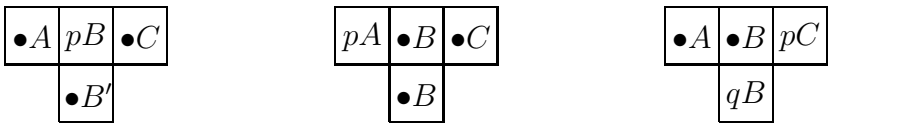


where $B \in \Sigma$ and $A, C \in \Sigma \cup \{[,]\}$.

We now need to describe how the cellular automaton simulates one computation step of the machine:



with $pA \to qA'+ \in \delta$, with $pC \to qC'+ \in \delta$,

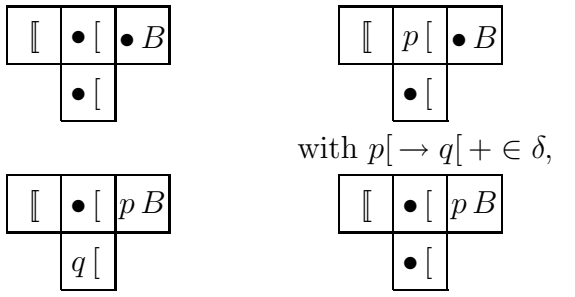


with $pB \to qB'\pm \in \delta$, with $pA \to qA'- \in \delta$, with $pC \to qC'- \in \delta$,

where $B \in \Gamma$ and $A, C \in \Gamma \cup \{[,]\}$.

We also have to deal with the border of the configuration, with the following

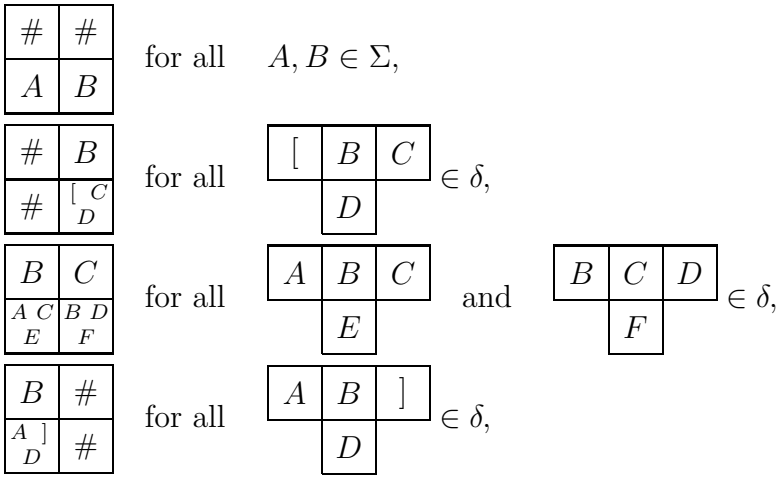
rules (only the left case is presented, the right case being symmetric):



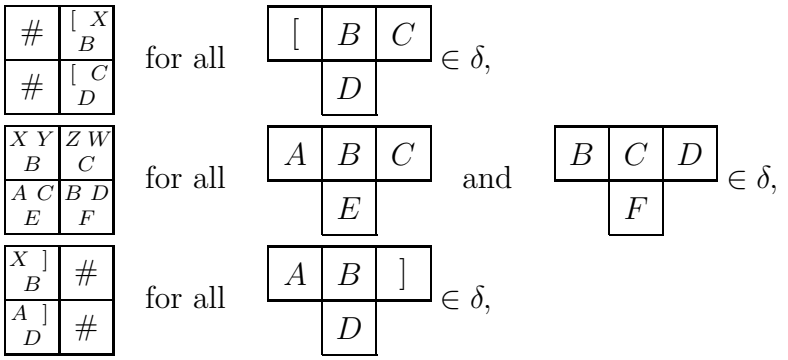
where $B \in \Gamma$. Contrary to the general case, it is now possible to stop the computation as soon as the current configuration c contains a state in F. Hence, as soon as any state $q_f \in F$ appears on the tape, we deterministically generate and propagate a special symbol $\perp$ until reaching an accepting configuration of the form $[\![\perp^*]\!]$. This will not be detailed here. Note that C is deterministic.

Let π be the morphism from $(\Gamma')^*$ to $(Q \cup \Gamma)^*$ defined by $\pi(\bullet, A) = A$ and $\pi(q, A) = qA$, it is straightforward to prove that there exists an run $c_0, c_1 \ldots c_n$ of C with c_n containing a symbol in F, if and only if the run $\pi(c_1), \ldots \pi(c_n)$ is accepting in M. It follows that the language recognised by C is $[L(M)]$. A deterministic cellular automaton C' recognising exactly $L(M)$ is easily obtained from C by extending the work alphabet.

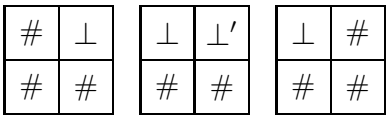
$3 \Rightarrow 4$: Let $C = (\Gamma, \Sigma, F, [,], \delta)$ be a deterministic cellular automaton. Let us construct an equivalent deterministic tiling system $S = (\Gamma', \Sigma, \#, \Delta)$. The set of symbols Γ' is equal to $\Sigma \cup \Gamma^3$. A symbol in Γ' is either a symbol in Σ or a tuple (l, n, r) of symbols in Γ where l (resp. r) represent the value of the left (resp. the right) neighbour at the previous step and c represent the current value of the cell. The set of tiles Δ contains the initial tiles:



as well as a set of tiles:



and finally, a set of tiles to detect an accepting configuration in $[F^+]$:



where $\perp$ and $\perp' \in \Gamma \times F \times \Gamma$.

It is easy to check that the tiling system S is deterministic. Let p be a picture in $P(S)$ and let $r_0, \ldots, r_m$ be the rows of $p_\#$, we show that the frontier of p is accepted by C. If we define the projection π by $\pi(\#w\#) = [w]$ for $w \in \Sigma^+$ and $\pi(\#(x_1, y_1, z_1) \ldots (x_n, y_n, z_n)\#) = [y_1 \ldots y_n]$, by construction $\pi(r_1), \ldots, \pi(r_{m-1})$ is an accepting run of C.

Conversely, if w is accepted by C there exists an accepting run $c_1, \ldots, c_n$. Let p be the $(n, |c_1| - 2)$-picture over Γ' defined by $p(1, i) = c_1(i + 1)$ for $i \in [2, |c_1| - 2]$ and $p(j, i) = (c_{j-1}(i - 2), c_j(i - 1), c_{j-1}(i))$ for $i \in [2, |c_1| - 2]$ and $j \in [2, n]$. It follows from the construction that $p_\# \in P(S)$ and hence $w \in L(S)$.

$4 \Rightarrow 2$: Let $S = (\Gamma, \Sigma, \#, \Delta)$ be a deterministic tiling system, we build a deterministic linear bounded Turing machine M recognising $L(S)$. The construction already given for the general case does not extend straightforwardly to the deterministic case, since computation may fail while trying to generate a possible new row from the existing one. In this case, we need to be able to ensure that the simulation does not fail unless the word is not accepted.

A row r in $\#\Gamma^+\# \cup \#^+$ is a successor of a row r' if $|r'| = |r|$ and the tiles of the picture p with rows r' and r are in Δ.

When starting with a word w, the machine M verifies that $\#w\#$ is a successor of $\#^{|w|+2}$ (if it is not, M rejects). The current row of M is now $\#w\#$ and M will deterministically compute the successor of this row. It enumerates all possible next rows starting with $\#^{|w|+2}$, checking for each of

them whether they are a successor of current row. If $\#^{|w|+2}$ is admissible, M accepts. If the current row does not admit a successor then M rejects. Otherwise, as S is deterministic there is exactly one successor r for the current row and M replaces the current row with r and repeat the procedure on r.

By construction, M accepts a word $w \in \Sigma^+$ if and only if $w \in L(S)$.

Conclusion

This thesis presents several contributions to the field of finitely presented infinite graphs. We give a brief summary of obtained results, as well as open questions and perspectives for each chapter of this document.

Summary of results and open questions

Term rewriting systems with a rational derivation

In Chapter 3, we studied three families of term-rewriting systems whose derivation relation is rational, and in particular can be described in a finite manner. They are the top-down, bottom-up and suffix rewriting systems. These results provide an immediate internal characterization of families of infinite graphs, as well as potential applications in the field of automatic verification of parameterized systems. Several open questions and leads for future research stem from this:

1. The derivations of our families of systems are defined as subfamilies of Raoult's rational term relations [Rao97]. Is it possible to provide direct characterizations of these families of relations using suitable families of tree automata or transducers?

2. What are the closure properties of each of these families of relations? In particular, do the derivations of suffix systems form a Boolean algebra?

3. How do the families of rewriting graphs of top-down, bottom-up and suffix systems compare to existing families? Can we provide other internal or external characterizations? What are the structural properties of these graphs (for instance, their traces)?

4. It is proved in [Cau00] that the derivations of left systems coincide with rational relations over words. Can we provide a characterization of Raoult's class of rational term relations as rewriting systems?

Infinite acceptors for context-sensitive languages

Chapter 4 provides an in-depth study of several families of infinite acceptors for deterministic and non-deterministic context-sensitive languages. In particular, Section 4.2 presents a new self-contained proof that the traces of rational graphs are the context-sensitive languages, even when restricting to finite degree graphs with a single initial vertex. We also characterized the traces of synchronized graphs of finite out-degree from a single vertex as the class of languages accepted by linearly bounded Turing machines in a linear number of head reversals, and proved that in the bounded-degree case they form a subfamily of deterministic context-sensitive languages. Finally, we proposed a condition over sets of transducers such that the traces of the corresponding rational graphs are precisely the deterministic context-sensitive languages.

In Section 4.3, we define the family of linearly bounded graphs, of which we give three different internal presentations. They are the Cayley-type graphs of length-decreasing rewriting systems, incremental context-sensitive transduction graphs and transition graphs of linearly bounded machines. A direct consequence of this last presentation is that the traces of these graphs are the context-sensitive languages. Other properties include the closure of this family under restriction to context-sensitive sets of vertices, and under restriction to reachable vertices. We finally investigated the case of deterministic context-sensitive languages and proved that they coincide with the traces of deterministic transition graphs of terminating linearly bounded machines.

This chapter is concluded by a comparison of both families of graphs up to isomorphism. We showed that every rational graph of bounded degree is a linearly bounded graph, but that the converse does not hold. Since bounded degree linearly bounded graphs accept all context-sensitive languages, this allows to define a hierarchy of bounded degree infinite graphs accepting each family of languages of the Chomsky hierarchy (from a single initial vertex).

The results presented in this chapter give rise to several questions and possible extensions.

1. Both linearly bounded graphs and rational graphs have an undecidable first-order theory. This is not the case for synchronized graphs, but it is not clear whether they accept all context-sensitive languages from a single initial vertex when their degree is finite and their sets of initial vertices are reduced to a single element. Is it possible to define a family of infinite graphs with a decidable first-order theory whose languages are all context-sensitive languages?

2. A possible way to answer this question would be to prove that finite-degree synchronized rational graphs indeed accept all context-sensitive languages

from a single vertex. This reduces to proving the equality of languages accepted by LBMs in a linear number of head reversals with context-sensitive languages, which is a non-trivial open question in complexity theory.

3. It seems feasible to extend most construction in Section 4.2 to the corresponding families of graphs over terms (term-automatic and term-rational graphs), using for instance Raoult's rational term transductions [Rao97].
4. The comparison of linearly bounded graphs with other families of graphs (like the above-mentioned families of graphs over terms) should be further investigated. Also, since all languages in the OI hierarchy are context-sensitive, it would be interesting to see whether the higher-order prefix-recognizable graphs are linearly bounded. A related question is to provide other (in particular external) presentations of these graphs.

Reachability analysis of higher-order context-free processes

Finally, we gave in Chapter 5 a preliminary reflexion on the extension of existing verification methods to the very expressive class of higher-order pushdown automata. We showed that the symbolic reachability analysis of [BEM97] can be extended to a sub-family of these systems, which we called higher-order context-free processes. They essentially correspond to higher-order pushdown automata with a single control state. Our technique uses regular word languages to represent sets of configurations, and computes (with a non-elementary time complexity) the set of predecessors of any such set by a given higher-order context-free process.

There are several remaining options to lead this study further. Independently from our work, Carayol defined in [Car05] a characterization of rational sets of higher-order stacks. This notion provides a natural symbolic forward and backward reachability analysis technique for the general class of higher-order pushdown automata. However, he also proves that most required operations (like the emptiness test for rational sets) are non-elementary. It would make sense to investigate in the light of these new results what complexity gain is obtained when restricting to higher-order context-free processes. More generally, it would be interesting to further investigate the structure of the family of transition graphs of higher-order context-free processes and to compare it with the general case. Finally, it would be interesting to understand whether higher-order context-free processes can be used as natural models for an interesting class of programs, in the way higher-order pushdown systems can be used as a translation for higher-order program schemes [Dam82, KNU02].

Concluding remarks

Our point of view is that the structural study of infinite graphs provides a general and elegant approach to the understanding of many different kinds of infinite-state systems. Seeing an infinite transition system as a graph may be considered as natural and useful as using finite graphs to represent the behaviour of finite-state machines. Results concerning alternative characterizations of infinite graphs may entail interesting consequences for each of their families of representatives, or provide simpler proofs of some of their properties.

Moreover, infinite graphs have many links to other areas of theoretical and applied computer science. They are very powerful modeling formalisms, and it can be hoped that this developing field of research will help provide new tools for the design of infinite-state verification methods. Moreover, infinite graphs seen as language acceptors shed a new light on some aspects of language theory, as pointed out for instance by Urvoy [Urv03], who established a precise link between families of infinite graphs and abstract families of languages (AFL).

Bibliography

[ABJ98] P. Abdulla, A. Bouajjani, and B. Jonsson. On-the-fly analysis of systems with unbounded, lossy fifo channels. In *Proceedings of the 10th International Conference on Computer Aided Verification (CAV 1998)*, volume 1427 of *Lecture Notes in Computer Science*, pages 305–318. Springer, 1998.

[AEM04] R. Alur, K. Etessami, and P. Madhusudan. A temporal logic of nested calls and returns. In *Proceedings of the 10th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2004)*, volume 2988 of *Lecture Notes in Computer Science*, pages 467–481. Springer, 2004.

[AJMD02] P. Abdulla, B. Jonsson, P. Mahata, and J. D'Orso. Regular tree model checking. In *Proceedings of the 14th International Conference on Computer Aided Verification (CAV 2002)*, volume 2404 of *Lecture Notes in Computer Science*, pages 555–568. Springer, 2002.

[AN82] A. Arnold and M. Nivat. Comportements de processus. In *Colloque de l'Association Francaise pour la Cybernétique Économique et Théorique: Les Mathématiques de l'Informatique (AFCET 1982)*, pages 35–68, 1982.

[Bar97] K. Barthelmann. On equational simple graphs. Technical Report 9, Universität Mainz, Institut für Informatik, 1997.

[Bar98] K. Barthelmann. When can an equational simple graph be generated by hyperedge replacement? In *Proceedings of the 23rd International Symposium on Mathematical Foundations of Computer Science (MFCS 1998)*, volume 1450 of *Lecture Notes in Computer Science*, pages 543–552. Springer, 1998.

[BC04] Y. Bertot and P. Castéran. *Coq'Art: The Calculus of Inductive Constructions*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2004.

[BCS96] O. Burkart, D. Caucal, and B. Steffen. Bisimulation collapse and the process taxonomy. In *Proceedings of the 7th International Conference on Concurrency Theory (CONCUR 1996)*, volume 1119 of *Lecture Notes in Computer Science*, pages 247–262. Springer, 1996.

[BEM97] A. Bouajjani, J. Esparza, and O. Maler. Reachability analysis of pushdown automata: Application to model-checking. In *Proceedings of the 8th International Conference on Concurrency Theory (CONCUR 1997)*, volume 1243 of *Lecture Notes in Computer Science*, pages 135–150. Springer, 1997.

[Ber79] J. Berstel. *Transductions and Context-Free Languages*. Leitfäden der angewandten Mathematik und Mechanik. Teubner, 1979.

[BG00] A. Blumensath and E. Grädel. Automatic structures. In *Proceedings of the 15th IEEE Symposium on Logic in Computer Science (LICS 2000)*, pages 51–62. IEEE, 2000.

[BGWW97] B. Boigelot, P. Godefroid, B. Willems, and P. Wolper. The power of qdds. In *Proceedings of the 4th International Symposium on Static Analysis (SAS 1997)*, volume 1302 of *Lecture Notes in Computer Science*, pages 172–186. Springer, 1997.

[BJNT00] A. Bouajjani, B. Jonsson, M. Nilsson, and T. Touili. Regular model checking. In *Proceedings of the 12th International Conference on Computer Aided Verification (CAV 2000)*, volume 1855 of *Lecture Notes in Computer Science*, pages 403–418. Springer, 2000.

[BK89] J. Bergstra and J. Klop. Process theory based on bisimulation semantics. In *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency (REX Workshop)*, volume 354 of *Lecture Notes in Computer Science*, pages 50–122. Springer, 1989.

[Blu01] A. Blumensath. Prefix-recognizable graphs and monadic second order logic. Technical Report AIB-06-2001, RWTH Aachen, 2001.

[BM04] A. Bouajjani and A. Meyer. Symbolic reachability analysis of higher-order context-free processes. In *Proceedings of the 24th International Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2004)*, volume 3328 of *Lecture Notes in Computer Science*, pages 135–147. Springer, 2004.

[Bou01] A. Bouajjani. Languages, rewriting systems, and verification of infinite-state systems. In *Proceedings of the 28th International Colloquium on Automata, Languages, and Programming (ICALP 2001)*,

volume 2076 of *Lecture Notes in Computer Science*, pages 24–39. Springer, 2001.

[BT02] A. Bouajjani and T. Touili. Extrapolating tree transformations. In *Proceedings of the 14th International Conference on Computer Aided Verification (CAV 2002)*, volume 2404 of *Lecture Notes in Computer Science*, pages 539–554. Springer, 2002.

[Büc62] J. Büchi. On a decision method in restricted second order arithmetic. In *Proceedings of the 1960 International Congress of Logic, Methodology and Philosophy of Science*, pages 1–12. Stanford University Press, 1962.

[Cac02] T. Cachat. Symbolic strategy synthesis for games on pushdown graphs. In *Proceedings of the 29th International Colloquium on Automata, Languages, and Programming (ICALP 2002)*, volume 2380 of *Lecture Notes in Computer Science*, pages 704–715. Springer, 2002.

[Car01] A. Carayol. Notions of determinism for rational graphs. Master's thesis, ENS Lyon, France, 2001.

[Car05] A. Carayol. Regular sets of higher-order pushdown stacks. In *Proceedings of the 30th International Symposium on Mathematical Foundations of Computer Science (MFCS 2005)*, Lecture Notes in Computer Science. Springer, 2005. To appear.

[Cau92] D. Caucal. On the regular structure of prefix rewriting. *Theoretical Computer Science*, 106:61–86, 1992.

[Cau95] D. Caucal. Bisimulation of context-free grammars and of pushdown automata. In A. Ponse, M. de Rijke, and Y. Venema, editors, *Modal Logic and Process Algebra*, volume 53 of *CSLI Lecture Notes*, pages 85–106. Stanford, 1995.

[Cau96] D. Caucal. On infinite transition graphs having a decidable monadic theory. In *Proceedings of the 23rd International Colloquium on Automata, Languages, and Programming (ICALP 1996)*, volume 1099 of *Lecture Notes in Computer Science*, pages 194–205. Springer, 1996.

[Cau00] D. Caucal. On word rewriting systems having a rational derivation. In *Proceedings of the 3rd International Conference on Foundations of Software Science and Computation Structures (FoSSaCS 2000)*,

volume 1784 of *Lecture Notes in Computer Science*, pages 48–62. Springer, 2000.

[Cau02] D. Caucal. On infinite terms having a decidable monadic theory. In *Proceedings of the 27th International Symposium on Mathematical Foundations of Computer Science (MFCS 2002)*, volume 2420 of *Lecture Notes in Computer Science*, pages 165–176. Springer, 2002.

[Cau03] D. Caucal. On the transition graphs of Turing machines. *Theoretical Computer Science*, 296:195–223, 2003.

[CC77] P. Cousot and R. Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Conference Record of the Fourth ACM Symposium on Principles of Programming Languages (POPL 1977)*, pages 238–252, 1977.

[CC03] A. Carayol and T. Colcombet. On equivalent representations of infinite structures. In *Proceedings of the 30th International Colloquium on Automata, Languages, and Programming (ICALP 2003)*, volume 2719 of *Lecture Notes in Computer Science*, pages 599–610. Springer, 2003.

[CDG+] H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications. Available on: http://www.grappa.univ-lille3.fr/tata.

[CES86] E. Clarke, A. Emerson, and P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 8(2):244–263, 1986.

[Cho59] N. Chomsky. On certain formal properties of grammars. *Information and Control*, 2:137–167, 1959.

[Cho82] L. Chottin. Langages algébriques et systèmes de réécriture rationnelle. *Informatique Théorique et Applications*, 16(2):93–112, 1982.

[Chu36] A. Church. A note on the entscheidungsproblem. *The Journal of Symbolic Logic*, 1(1):40–41, 1936.

[CK98] H. Calbrix and T. Knapik. A string-rewriting characterization of Muller and Schupp's context-free graphs. In *Proceedings of the 18th*

International Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 1998), volume 1530 of *Lecture Notes in Computer Science*, pages 331–342. Springer, 1998.

[CK01] D. Caucal and T. Knapik. An internal presentation of regular graphs by prefix-recognizable graphs. *Theoretical Computer Science*, 34:299–336, 2001.

[CK02a] D. Caucal and T. Knapik. A Chomsky-like hierarchy of infinite graphs. In *Proceedings of the 27th International Symposium on Mathematical Foundations of Computer Science (MFCS 2002)*, volume 2420 of *Lecture Notes in Computer Science*, pages 177–187. Springer, 2002.

[CK02b] B. Courcelle and T. Knapik. The evaluation of first-order substitution is monadic second-order compatible. *Theoretical Computer Science*, 281(1-2):177–206, 2002.

[CM05a] A. Carayol and A. Meyer. Context-sensitive languages, rational graphs and determinism. To appear (preliminary version available at `http://www.liafa.jussieu.fr/~ameyer/`), 2005.

[CM05b] A. Carayol and A. Meyer. Linearly bounded infinite graphs. In *Proceedings of the 30th International Symposium on Mathematical Foundations of Computer Science (MFCS 2005)*, Lecture Notes in Computer Science. Springer, 2005. To appear.

[CM05c] A. Carayol and A. Meyer. Linearly bounded infinite graphs (long version). Submitted (preliminary version available at `http://www.liafa.jussieu.fr/~ameyer/`), 2005.

[Col02] T. Colcombet. On families of graphs having a decidable first order theory with reachability. In *Proceedings of the 29th International Colloquium on Automata, Languages, and Programming (ICALP 2002)*, volume 2380 of *Lecture Notes in Computer Science*, pages 98–109. Springer, 2002.

[Col04] T. Colcombet. *Représentations et propriétés de structures infinies*. PhD thesis, Université de Rennes 1, France, 2004.

[Cou89] B. Courcelle. The monadic second-order logic of graphs, II: Infinite graphs of bounded width. *Mathematical System Theory*, 21:187–221, 1989.

[Cou90] B. Courcelle. Graph rewriting: An algebraic and logic approach. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics*, pages 193–242. Elsevier, 1990.

[Cou94] B. Courcelle. Monadic second-order definable graph transductions: A survey. *Theoretical Computer Science*, 126(1):53–75, 1994.

[Cou97] B. Courcelle. The expression of graph properties and graph transformations in monadic second-order logic. In G. Rozenberg, editor, *Handbook of Graph Grammars and Computing by Graph Transformation. Vol. I: Foundations*, chapter 5, pages 313–400. World Scientific, 1997.

[CW03] A. Carayol and S. Wöhrle. The caucal hierarchy of infinite graphs in terms of logic and higher-order pushdown automata. In *Proceedings of the 23rd International Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2003)*, volume 2914 of *Lecture Notes in Computer Science*, pages 112–123. Springer, 2003.

[Dam82] W. Damm. The IO- and OI-hierarchies. *Theoretical Computer Science*, 20:95–207, 1982.

[DHLT90] M. Dauchet, T. Heuillard, P. Lescanne, and S. Tison. Decidability of the confluence of finite ground term rewrite systems and of other related term rewrite systems. *Information and Computation*, 88(2):187–201, 1990.

[DJ90] N. Dershowitz and J.-P. Jouannaud. Rewrite systems. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science, Vol. B*, pages 243–320. Elsevier, 1990.

[DT85] M. Dauchet and S. Tison. Decidability of the confluence of finite ground term rewrite systems. In *Proceedings of the 5th International Symposium on Fundamentals of Computation Theory, (FCT 1985)*, volume 199 of *Lecture Notes in Computer Science*, pages 80–89. Springer, 1985.

[DTHL87] M. Dauchet, S. Tison, T. Heuillard, and P. Lescanne. Decidability of the confluence of ground term rewriting systems. In *Proceedings of the 2nd IEEE Symposium on Logic in Computer Science (LICS 1987)*, pages 353–359. IEEE, 1987.

[EHRS00] J. Esparza, D. Hansel, P. Rossmanith, and S. Schwoon. Efficient algorithm for model checking pushdown systems. In *Proceedings of the 12th International Conference on Computer Aided Verification (CAV 2000)*, volume 1885 of *Lecture Notes in Computer Science*, pages 232–247. Springer, 2000.

[EM65] C. Elgot and J. Mezei. On relations defined by finite automata. *IBM Journal of Research and Development*, 9:47–68, 1965.

[Eme90] E. A. Emerson. Temporal and modal logic. In *Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics (B)*, pages 995–1072. Elsevier, 1990.

[Eng83] J. Engelfriet. Iterated pushdown automata and complexity classes. In *Proceedings of the 15th Annual ACM Symposium on Theory of Computing (STOC 1983)*, pages 365–373. ACM, ACM Press, 1983.

[EW05] Z. Esik and P. Weil. Algebraic recognizability of regular tree languages. *Theoretical Computer Science*, 2005. To appear.

[For00] L. Fortnow. Diagonalization. *Bulletin of the European Association for Theoretical Computer Science*, 71:102–112, 2000.

[FS93] C. Frougny and J. Sakarovitch. Synchronized rational relations of finite and infinite words. *Theoretical Computer Science*, 108(1):45–82, 1993.

[GR96] D. Giammarresi and A. Restivo. *Handbook of Formal Languages*, volume 3, chapter Two-dimensional languages. Springer, 1996.

[GS97] F. Gécseg and M. Steinby. *Handbook of Formal Languages*, volume 3, chapter 1, pages 1–68. Springer, 1997.

[GV98] P. Gyenizse and S. Vágvölgyi. Linear generalized semi-monadic rewrite systems effectively preserve recognizability. *TCS*, 194(1–2):87–122, 1998.

[HJJ+95] J. Henriksen, J. Jensen, M. Jørgensen, N. Klarlund, R. Paige, T. Rauhe, and A. Sandholm. Mona: Monadic second-order logic in practice. In *Proceedings of the 1st International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 1995)*, volume 1019 of *Lecture Notes in Computer Science*, pages 89–110. Springer, 1995.

[HU79] J. Hopcroft and J. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, 1979.

[Imm88] N. Immerman. Nondeterministic space is closed under complementation. *SIAM Journal on Computing*, 17(5):935–938, 1988.

[JW96] D. Janin and I. Walukiewicz. On the expressive completeness of the propositional mu-calculus with respect to monadic second order logic. In *Proceedings of the 7th International Conference on Concurrency Theory (CONCUR 1996)*, volume 1119 of *Lecture Notes in Computer Science*, pages 263–277. Springer, 1996.

[Kle56] S. Kleene. Representation of events in nerve nets and finte automata. In *Automata studies*, volume 34 of *Annals of Mathematics Studies*, pages 3–40. Princeton University Press, 1956.

[KMM+97] Y. Kesten, O. Maler, M. Marcus, A. Pnueli, and E. Shahar. Symbolic model checking with rich assertional languages. In *Proceedings of the 9th International Conference on Computer Aided Verification (CAV 1997)*, volume 1254 of *Lecture Notes in Computer Science*, pages 424–435. Springer, 1997.

[Kni64] J. D. Mc Knight. Kleene quotient theorem. *Pacific Journal of Mathematics*, pages 1343–1352, 1964.

[KNU02] T. Knapik, D. Niwinski, and P. Urzyczyn. Higher-order pushdown trees are easy. In *Proceedings of the 5th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS 2002)*, volume 2303 of *Lecture Notes in Computer Science*, pages 205–222. Springer, 2002.

[Kob69] K. Kobayashi. Classification of formal languages by functional binary transductions. *Information and Control*, 15(1):95–109, 1969.

[KP99] T. Knapik and É. Payet. Synchronized product of linear bounded machines. In *Proceedings of the 12th International Symposium on Fundamentals of Computation Theory, (FCT 1999)*, volume 1684 of *Lecture Notes in Computer Science*, pages 362–373. Springer, 1999.

[Kur64] S. Kuroda. Classes of languages and linear-bounded automata. *Information and Control*, 7(2):207–223, 1964.

[Löd02] Christof Löding. Ground tree rewriting graphs of bounded tree width. In *Proceedings of the 19th Annual Symposium on Theoretical*

Aspects of Computer Science (STACS 2002), volume 2285 of *Lecture Notes in Computer Science*, pages 559–570. Springer, 2002.

[LS97] M. Latteux and D. Simplot. Context-sensitive string languages and recognizable picture languages. *Information and Computation*, 138(2):160–169, 1997.

[LST98] M. Latteux, D. Simplot, and A. Terlutte. Iterated length-preserving rational transductions. In *Proceedings of the 23rd International Symposium on Mathematical Foundations of Computer Science (MFCS 1998)*, volume 1450 of *Lecture Notes in Computer Science*, pages 286–295. Springer, 1998.

[LW01] K. Lodaya and P. Weil. Rationality in algebras with a series operation. *Information and Computation*, 171(2):269–293, 2001.

[MD98] J. Mazoyer and M. Delorme. Cellular automata: a parallel model. In *Cellular Automata as Language Recognizers*, pages 153–180. Kluwer Academic Publishings, 1998.

[Mey02] A. Meyer. Sur des sous-familles de graphes rationnels. Master's thesis, Université de Rennes 1, France, 2002.

[Mey04] A. Meyer. On term rewriting systems having a rational derivation. In *Proceedings of the 7th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS 2004)*, volume 2987 of *Lecture Notes in Computer Science*, pages 378–392. Springer, 2004.

[Mil89] R. Milner. *Communication and Concurrency*. Prentice Hall International, 1989.

[Mor00] C. Morvan. On rational graphs. In *Proceedings of the 3rd International Conference on Foundations of Software Science and Computation Structures (FoSSaCS 2000)*, volume 1784 of *Lecture Notes in Computer Science*, pages 252–266. Springer, 2000.

[Mor01] C. Morvan. *Les graphes rationnels*. PhD thesis, Université de Rennes 1, France, 2001.

[MR05] Christophe Morvan and Chloé Rispal. Families of automata characterizing context-sensitive languages. *Acta Informatica*, 41(4-5):293–314, 2005.

[MS85] D. E. Muller and P. E. Schupp. The theory of ends, pushdown automata, and second-order logic. *Theoretical Computer Science*, 37:51–75, 1985.

[MS01] C. Morvan and C. Stirling. Rational graphs trace context-sensitive languages. In *Proceedings of the 26th International Symposium on Mathematical Foundations of Computer Science (MFCS 2001)*, volume 2136 of *Lecture Notes in Computer Science*, pages 548–559. Springer, 2001.

[Mye79] G. Myers. *The Art of Software Testing*. John Wiley & Sons, 1979.

[Nec97] G. Necula. Proof-carrying code. In *Proceedings of the 24th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL 1997)*, pages 106–119. ACM Press, 1997.

[NPW02] T. Nipkow, L. Paulson, and M. Wenzel. *Isabelle/HOL - A Proof Assistant for Higher-Order Logic*. Springer, 2002.

[ORS92] S. Owre, J. Rushby, , and N. Shankar. PVS: A prototype verification system. In *Proceedings of the 11th International Conference on Automated Deduction (CADE 1992)*, volume 607 of *Lecture Notes in Artificial Intelligence*, pages 748–752. Springer, 1992.

[Pay00] É. Payet. *Thue Specifications, Infinite Graphs and Synchronized Product*. PhD thesis, Université de la Réunion, 2000.

[Pen74] M. Penttonen. One-sided and two-sided context in formal grammars. *Information and Control*, 25(4):371–392, 1974.

[Pnu77] A. Pnueli. The temporal logic of programs. In *Proceedings of the 18th IEEE Symposium on the Foundations of Computer Science (FOCS 1977)*, pages 46–57. IEEE Computer Society Press, 1977.

[Rab69] M. Rabin. Decidability of second-order theories and automata on infitite trees. *Trans. of Amer. Math. Soc.*, 141:1–35, 1969.

[Rao91] J.-C. Raoult. A survey of tree transductions. Technical Report 1410, Inria, April 1991.

[Rao97] J.-C. Raoult. Rational tree relations. *Bulletin of the Belgian Mathematics Society*, 4:149–176, 1997.

[Ris02] C. Rispal. The synchronized graphs trace the context-sensitive languages. In *Proceedings of the 4th International Workshop on Verification of Infinite-State Systems (INFINITY 2002)*, volume 68 of *Electronic Notes in Theoretical Computer Science*, 2002.

[Sak03] J. Sakarovitch. *Éléments de théorie des automates*. Éditions Vuibert, 2003.

[Sti98] C. Stirling. The joys of bisimulation. In *Proceedings of the 23rd International Symposium on Mathematical Foundations of Computer Science (MFCS 1998)*, volume 1450 of *Lecture Notes in Computer Science*, pages 142–151. Springer, 1998.

[Sti00] C. Stirling. Decidability of bisimulation equivalence for pushdown processes. Technical Report EDI-INF-RR-0005, School of Informatics, University of Edinburgh, 2000.

[TKS00] T. Takai, Y. Kaji, and H. Seki. Right-linear finite path overlapping term rewriting systems effectively preserve recognizability. In *Proceedings of the 11th International Conference on Rewriting Techniques and Applications (RTA 2000)*, volume 1833 of *Lecture Notes in Computer Science*, pages 246–260. Springer, 2000.

[Tra50] B. Trakhtenbrot. The impossibilty of an algorithm for the decision problem for finite models. *Doklady Akademii Nauk SSR*, 70:569–572, 1950.

[Tur36] A. Turing. On computable numbers, with an application to the entscheidungsproblem. *Proceedings of the London Mathematical Society*, 42:230–265, 1936.

[Urv03] T. Urvoy. *Familles abstraites de graphes*. PhD thesis, Université de Rennes 1, France, June 2003.

[vM04] D. van Melkebeek. Time-space lower bounds for NP-complete problems. *Current Trends in Theoretical Computer Science*, pages 265–291, 2004.

[VW86] M. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification (preliminary report). In *Proceedings of the 1st Symposium on Logic in Computer Science (LICS 1986)*, pages 332–344. IEEE Computer Society, 1986.

[Wal96] I. Walukiewicz. Pushdown processes: Games and model checking. In *Proceedings of the 8th International Conference on Computer Aided Verification (CAV 1996)*, volume 1102 of *Lecture Notes in Computer Science*, pages 62–74. Springer, 1996.

[Wal02] I. Walukiewicz. Monadic second-order logic on tree-like structures. *Theoretical Computer Science*, 275(1–2):311–346, 2002.

[WB98] P. Wolper and B. Boigelot. Verifying systems with infinite but regular state spaces. In *Proceedings of the 10th International Conference on Computer Aided Verification (CAV 1998)*, volume 1427 of *Lecture Notes in Computer Science*, pages 88–97. Springer, 1998.

[Web96] A. Weber. Decomposing a k-valued transducer into k unambiguous ones. *Informatique Théorique et Applications*, 30(5):379–413, 1996.

[Wei04] P. Weil. Algebraic recognizability of languages. In *Proceedings of the 29th International Symposium on Mathematical Foundations of Computer Science (MFCS 2004)*, volume 3153 of *Lecture Notes in Computer Science*, pages 149–175. Springer, 2004.

Résumé

Cette thèse s'inscrit dans l'étude de familles de graphes infinis de présentation finie, de leurs propriétés structurelles, ainsi que des comparaisons entre ces familles. Étant donné un alphabet fini Σ, un graphe infini étiqueté par Σ peut être caractérisé par un ensemble fini de relations binaires $(R_a)_{a\in\Sigma}$ sur un domaine dénombrable V quelconque. De multiples caractérisations finies de tels ensembles de relations existent, soit de façon explicite grâce à des systèmes de réécriture ou à divers formalismes de la théorie des automates, soit de façon implicite.

Après un survol des principaux résultats existants, nous nous intéressons plus particulièrement à trois problèmes. Dans un premier temps, nous définissons trois familles de systèmes de réécriture de termes dont nous démontrons que la relation de dérivation peut être représentée de façon finie. De ces résultats découlent plusieurs questions sur les familles de graphes infinis correspondantes. Dans un second temps, nous étudions deux familles de graphes dont les ensembles de traces forment la famille des langages contextuels, à savoir les graphes rationnels et les graphes linéairement bornés. Nous nous intéressons en particulier au cas des langages contextuels déterministes, ainsi qu'à la comparaison structurelle de ces deux familles. Enfin, d'un point de vue plus proche du domaine de la vérification, nous proposons un algorithme de calcul des prédécesseurs pour une famille d'automates à pile d'ordre supérieur.

Abstract

This thesis contributes to the study of families of finitely presented infinite graphs, their structural properties and their relations to each other. Given a finite alphabet Σ, a Σ-labeled infinite graph can be characterized as a finite set of binary relations $(R_a)_{a\in\Sigma}$ over an arbitrary countable domain V. There are many ways to finitely characterize such sets of relations, either explicitly using rewriting systems or formalisms from automata theory, either externally.

After giving an overview of the main results in this domain, we focus on three specific problems. In a first time, we define several families of term-rewriting systems whose derivation relation can be finitely represented. These results raise interesting questions concerning the corresponding families of infinite graphs. In a second time, we study two families of infinite graphs whose sets of traces (or languages) coincide with the well-known family of context-sensitive languages. They are the rational graphs and the linearly bounded graphs. We investigate the case of deterministic context-sensitive languages, and establish a structural comparison between these two families of graphs. Finally, in an approach closer to the concerns of the verification community, we propose a symbolic reachability algorithm for a class of higher-order pushdown automata.